



Calhoun: The NPS Institutional Archive
DSpace Repository

Theses and Dissertations

1. Thesis and Dissertation Collection, all items

2020-03

CROSS-DOMAIN IDENTIFICATION OF ROAD NETWORKS USING DOMAIN-ADAPTED CONVOLUTIONAL NEURAL NETWORKS

Peterson, Teal A.

Monterey, CA; Naval Postgraduate School

<http://hdl.handle.net/10945/64901>

Downloaded from NPS Archive: Calhoun



Calhoun is a project of the Dudley Knox Library at NPS, furthering the precepts and goals of open government and government transparency. All information contained herein has been approved for release by the NPS Public Affairs Officer.

Dudley Knox Library / Naval Postgraduate School
411 Dyer Road / 1 University Circle
Monterey, California USA 93943

<http://www.nps.edu/library>



NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

**CROSS-DOMAIN IDENTIFICATION OF ROAD
NETWORKS USING DOMAIN-ADAPTED
CONVOLUTIONAL NEURAL NETWORKS**

by

Teal A. Peterson

March 2020

Thesis Advisor:

Douglas P. Horner

Co-Advisor:

Geoffrey G. Xie

Second Reader:

Michael R. McCarrin

Approved for public release. Distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			<i>Form Approved OMB No. 0704-0188</i>	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE March 2020	3. REPORT TYPE AND DATES COVERED Master's thesis	
4. TITLE AND SUBTITLE CROSS-DOMAIN IDENTIFICATION OF ROAD NETWORKS USING DOMAIN-ADAPTED CONVOLUTIONAL NEURAL NETWORKS			5. FUNDING NUMBERS	
6. AUTHOR(S) Teal A. Peterson				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) SPAWAR			10. SPONSORING / MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.			12b. DISTRIBUTION CODE A	
13. ABSTRACT (maximum 200 words) Convolutional neural networks (CNNs) are a powerful tool for identification of patterns and objects within imagery or video. Training CNNs that can generalize well to their intended target dataset can require large amounts of labeled source data. The characteristics and distribution of this source (training) data must be representative of the target dataset for it to perform well. Labeled source data that fits this requirement is not always readily available. Research published by Ganin et al., in a 2016 paper titled "Domain-Adversarial Training of Neural Networks," demonstrates that CNNs trained on a labeled source dataset can be adapted to generalize well to a target dataset through a process called domain adaption. In their research, they show that domain-adversarial neural networks (DANNs) improve performance on their target dataset relative to non-adapted CNNs. The purpose of this research is to explore the ability of DANNs to improve unmanned aerial vehicle (UAV) onboard classification of objects by adapting a CNN trained on satellite imagery to UAV aerial imagery. We show that DANNs do improve performance for this use case using several DANN architectures and datasets. This furthers other Naval Postgraduate School research efforts into autonomous UAV navigation and identification of targets of interest.				
14. SUBJECT TERMS neural networks, deep learning, convolutional neural network, artificial neural networks, domain adaptation, domain divergence, domain adversarial neural network, gradient descent, gradient reversal layer, H divergence, proxy A distance, back propagation, data, Scan Eagle, ArcGIS, geographic information system, video, imagery, remote sensing, satellite, unmanned aerial vehicle, unmanned aerial system, computer vision, artificial intelligence, machine learning, autonomy			15. NUMBER OF PAGES 133	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UU	

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

**CROSS-DOMAIN IDENTIFICATION OF ROAD NETWORKS USING
DOMAIN-ADAPTED CONVOLUTIONAL NEURAL NETWORKS**

Teal A. Peterson
Major, United States Marine Corps
BS, University of California - Davis, 2008

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

**NAVAL POSTGRADUATE SCHOOL
March 2020**

Approved by: Douglas P. Horner
Advisor

Geoffrey G. Xie
Co-Advisor

Michael R. McCarrin
Second Reader

Peter J. Denning
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Convolutional neural networks (CNNs) are a powerful tool for identification of patterns and objects within imagery or video. Training CNNs that can generalize well to their intended target dataset can require large amounts of labeled source data. The characteristics and distribution of this source (training) data must be representative of the target dataset for it to perform well. Labeled source data that fits this requirement is not always readily available. Research published by Ganin et al., in a 2016 paper titled “Domain-Adversarial Training of Neural Networks,” demonstrates that CNNs trained on a labeled source dataset can be adapted to generalize well to a target dataset through a process called domain adaption. In their research, they show that domain-adversarial neural networks (DANNs) improve performance on their target dataset relative to non-adapted CNNs. The purpose of this research is to explore the ability of DANNs to improve unmanned aerial vehicle (UAV) onboard classification of objects by adapting a CNN trained on satellite imagery to UAV aerial imagery. We show that DANNs do improve performance for this use case using several DANN architectures and datasets. This furthers other Naval Postgraduate School research efforts into autonomous UAV navigation and identification of targets of interest.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

1	Introduction	1
1.1	Problem Statement	3
1.2	Research Questions	4
1.3	Overview	5
1.4	Significance to the Field	5
2	Background	9
2.1	Introduction	9
2.2	Machine Learning	9
2.3	Remote Sensing	19
2.4	Summary	29
3	Methodology	31
3.1	Introduction	31
3.2	Domain Adaptation	31
3.3	Domain Divergence	34
3.4	DANNs	41
3.5	Summary	46
4	Design	49
4.1	Introduction	49
4.2	Setting	49
4.3	Experimentation	50
4.4	Materials	51
4.5	Data	55
4.6	Procedures	57
4.7	Data Analysis and Measurements of Performance	62
4.8	Summary	63

5	Results	69
5.1	Introduction	69
5.2	Phase I – Define Domains and Build Datasets	69
5.3	Phase II – Build and Train Source-Domain-Only CNNs	70
5.4	Phase III – Build and Train Target-Domain-Only CNNs	71
5.5	Phase IV – Evaluate CNNs on Source and Target Domain Test Data	72
5.6	Phase V – Build, Train, and Evaluate DANNs	72
5.7	Summary	73
6	Conclusions, Limitations, and Future Work	101
6.1	Introduction	101
6.2	Conclusions	101
6.3	Limitations and Future Work.	103
6.4	Summary	108
	List of References	109
	Initial Distribution List	113

List of Figures

Figure 1.1	Thesis phases, goals, and data-flow.	6
Figure 1.2	CNN by phase.	8
Figure 2.1	Non-adapted CNN.	12
Figure 2.2	Non-adapted CNN labeling cross-domain.	13
Figure 2.3	An artificial neuron compared to an actual neuron.	14
Figure 2.4	Chain Rule.	18
Figure 2.5	Spectral reflectance curves.	20
Figure 2.6	Composite satellite images.	21
Figure 2.7	Ground sample distance (GSD) illustration.	22
Figure 2.8	Elements of image interpretation.	25
Figure 3.1	Notional domains.	33
Figure 3.2	Domain adaptation.	35
Figure 3.3	Domain classifier.	43
Figure 3.4	Sample MNIST/MNIST-M clips.	46
Figure 3.5	MNIST/MNIST-M DANN.	47
Figure 4.1	Camp Roberts region.	64
Figure 4.2	Model building blocks.	65
Figure 4.3	Simple CNN architecture.	66
Figure 4.4	Small Inception/ResNetV2-ish CNN architecture.	67
Figure 4.5	CNN architecture using frozen Inception/ResNetV2 base.	68

Figure 5.1	Sample 160x160 “road” clips.	76
Figure 5.2	Training and validation for Simple CNN on MNIST data.	77
Figure 5.3	Training and validation for Custom CNN on MNIST data.	78
Figure 5.4	Training and validation for CNN with frozen feature extractor on MNIST data.	79
Figure 5.5	Training and validation for Simple CNN on 160x160 satellite clips.	80
Figure 5.6	Training and validation for Custom CNN on 160x160 satellite clips.	81
Figure 5.7	Training and validation for CNN with frozen feature extractor on 160x160 satellite clips.	82
Figure 5.8	Training and validation for Simple CNN on 120x120 satellite clips.	83
Figure 5.9	Training and validation for Custom CNN on 120x120 satellite clips.	84
Figure 5.10	Training and validation for CNN with frozen feature extractor on 120x120 satellite clips.	85
Figure 5.11	Training and validation for the Simple CNN on MNIST-M data.	86
Figure 5.12	Training and validation for the Custom CNN on MNIST-M data.	87
Figure 5.13	Training and validation for the CNN with frozen feature extractor on MNIST-M data.	88
Figure 5.14	Training and validation for the Simple CNN on 40X40 unmanned aerial system (UAS) video clips.	89
Figure 5.15	Training and validation for the Custom CNN on 40X40 UAS video clips.	90
Figure 5.16	Training and validation for the CNN with frozen feature extractor on 40X40 UAS video clips.	91
Figure 5.17	Training accuracy for Simple DANN on MNIST/MNIST-M data.	93
Figure 5.18	Training accuracy for Custom DANN on MNIST/MNIST-M data.	94
Figure 5.19	Training and validation accuracy for DANN with frozen feature extractor on MNIST/MNIST-M data.	94

Figure 5.20	Training accuracy for Simple DANN on 120x120/40x40 data. . .	96
Figure 5.21	Training accuracy for Custom DANN on 120x120/40x40 data. . .	97
Figure 5.22	Training accuracy for Simple DANN on 160x160/40x40 data. . .	98
Figure 5.23	Training accuracy for Custom DANN on 160x160/40x40 data. . .	99
Figure 6.1	Jeep trails in grass.	104

THIS PAGE INTENTIONALLY LEFT BLANK

List of Tables

Table 4.1	Satellite imagery metadata.	56
Table 4.2	Domain class definitions.	59
Table 4.3	Non-DANN architectures	60
Table 5.1	Satellite imagery datasets	70
Table 5.2	UAS video dataset	70
Table 5.3	Source-domain only CNN Accuracy	71
Table 5.4	Target-domain only CNN Accuracy	72
Table 5.5	Evaluation on MNIST/MNIST-M CNN	73
Table 5.6	Evaluation on 120x120 trained CNN	74
Table 5.7	Evaluation on 160x160 trained CNN	75
Table 5.8	Evaluation of MNIST/MNIST-M trained DANN on MNIST-M test dataset	92
Table 5.9	Evaluation of 120x120/40x40 trained DANNs on 40x40 test dataset	95
Table 5.10	Evaluation of 160x160/40x40 trained DANNs on 40x40 test dataset	95

THIS PAGE INTENTIONALLY LEFT BLANK

List of Acronyms and Abbreviations

AGL	Above Ground Level
AI	Artificial Intelligence
CAVR	Center for Autonomous Vehicle Research
CNN	Convolutional Neural Network
COMTHIRDFLT	Commander, U.S. Third Fleet
CRUSER	Consortium for Robotics and Unmanned Systems Education and Research
DA	Domain Adaption
DANN	Domain-Adversarial Neural Network
DoD	Department of Defense
EV-WHS	EnhancedView Web Hosting Service
FOV	Field of View
GCS	Ground Control Station
GIS	Geographic Information System
GPS	Global Positioning System
GRL	Gradient Reversal Layer
GSD	Ground Sample Distance
HSV	Hue, Saturation, Value
IFOV	Instantaneous Field of View
IR	Infrared

ML	Machine Learning
MSL	Mean Sea Level
MTX	Multi-Threaded Experimentation
NAVSPECWAR	Naval Special Warfare Command
NAVWAR	Naval Information Warfare Systems Command
NGA	National Geospatial-Intelligence Agency
NN	Neural Network
NPS	Naval Postgraduate School
PCAP	Packet Capture
ReLU	Rectified Linear Unit
RGB	Red, Green, Blue
ROS	Robot Operating System
SEAL	U.S. Navy (USN) Sea, Air, and Land
SGD	Stochastic Gradient Descent
UAS	Unmanned Aerial System
UAV	Unmanned Aerial Vehicle
USG	United States Government
USN	U.S. Navy
VC	Vapnik-Chervonenkis
WV02	WorldView-2

Acknowledgments

Family is often forgotten despite their importance to supporting an endeavor like this. I am grateful to my children and family for their enduring love and patience through long hours of reading, typing, and frustrating troubleshooting. I see them as the reason for my continued service and this effort, despite career priorities that often communicate anything but. Keane and Echo, thank you for trusting me on this and believing in something beyond yourselves. You both are amazing kids; I could not have asked for better.

Dr. Horner, Dr. Xie, and Prof. McCarrin: thank you for letting my ambitions take this thesis where it did and for the steady nudges that kept it progressing despite all of my personal distractions over the past year. Thanks to you, this thesis is a product I am proud to have my name on and a product well beyond the “check in the box” it often becomes for other students. I look forward to continuing work on this with you even as I move on to other ventures.

During my time at Naval Postgraduate School (NPS), I had the good fortune of bumping into and temporarily working for Col Brandel at DIU. I appreciate the mentorship, approachability, and opportunities you’ve introduced me to. You’ve personally renewed my belief that one person really can have an outsized impact. I’m fortunate to have your continued guidance and look forward to where it may take me.

I am also very appreciative of Col Lyons, Col Pugh, and LtCol Forbell at NPS for going to bat for me more than once over this past year. I am forever grateful for the knee you allowed me to take and the hand you extended to help me back up afterwards. It seemed an uncommon grace you proved can absolutely have a place in our organization. Your compassion has been an experience I will never forget and an example I will always strive to emulate.

Finally, I’ve got to throw some major thanks toward Mr. Gregg Kendrick, LtCol Craig Clarkson, and Maj Joseph Plot. Through their quick work, gracious empathy, and advocacy I feel positioned to run harder than ever before. I’m indebted to you for the hand you extended when others were more than willing to leave me on the field. I look forward to seeing just how much I accomplish for and with you over the next few years.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:

Introduction

Currently, unmanned aerial systems (UASs) provide a plethora of sensor data to the warfighter, which enhances situational awareness and speeds accurate decision making. The process of turning this data into actionable intelligence, however, requires substantial support, resource overhead, and delay in the form of UAS operators; large, reliable data transmissions; and computer processing power away from the objective area. Human error is inherent in this process. Looking towards an environment where small electro-optical footprints and fast op-tempo are desired, there is a need for autonomous systems that can operate locally, with limited delay in the observe-orient-decide-act production of usable, accurate information for the warfighter.

Researchers continue to search for ways to conduct off-UAS processing of collected imagery using machine learning and neural networks. While this research is expected to speed analysis of collected UAS data, it does not influence control of the UAS or reduce the amount of data that it must transmit. Research conducted for this thesis seeks to help solve some of these problems by helping move image processing and UAS control to the edge. Accomplishing this will relieve the warfighter from ‘one-more’ system that they are required to operate.

Efforts to accomplish this have been ongoing within Consortium for Robotics and Unmanned Systems Education and Research (CRUSER). In November 2017, a CRUSER-supported initiative called Multi-Threaded Experimentation (MTX) brought together Naval Special Warfare Command (NAVSPECWAR), Commander, U.S. Third Fleet (COMTHIRDFLT), Naval Postgraduate School (NPS) and Naval Information Warfare Systems Command (NAVWAR) together for three weeks at San Clemente Island. The test was run by Dr. Doug Horner (the primary advisor for this research) and included three unmanned aerial vehicles (UAVs) (ScanEagles), two unmanned surface vehicle (SeaFoxes), and two autonomous underwater vehicles (REMUS) in the development and experimentation of a UxV Network Control System. All systems had wireless mesh communications and supported a U.S. Navy (USN) Sea, Air, and Land (SEAL) element going ashore with a USN ship acting as command and control.

Part of this experiment included flying the ScanEagles with a secondary controller. The UAVs flew optimal trajectories (relative to the transiting SEALs) to provide an optical early warning system for opposing forces. This resulted in a desire to include an autonomous road detection component which would provide the ScanEagle the ability to independently create a more accurate picture of its operating environment. Increasing the ability of the ScanEagle to understand its environment opens opportunities for reducing task loading for the teams employing them and increases the services that the ScanEagle could provide. This might include more efficient search for threats or autonomous route planning for the mobile team to avoid detection.

The video output from the ScanEagle is typically what is used by the ground operator for detecting threats and route planning. Automating this task will likely require near-human classification performance. Today, convolutional neural networks (CNNs) are the only tool available that has successfully reached this level of success. We seek to explore the applicability of CNNs to the task of identifying roads from on-board the ScanEagle. Training CNNs to accomplish this will require a substantial amount of labeled ScanEagle video footage, or some other means of generalizing a CNNs to this task.

CNNs that produce high classification accuracy require an abundance of labeled data during training. The importance of large amounts of data for training cannot be overstated. During training, CNNs learn the features that define the characteristics which help it “decide” its output values. If the training dataset is not representative of the true dataset, then the CNN will not output values representative of the true dataset. For example, assume you have a bag containing one million red and blue cubes and rods. To train a model that can identify shape, you might pull out only ten of your shapes and label them. By chance, this random draw consists of only red cubes and blue rods. You train your model on this random draw, then begin to test this model on an additional draw of ten shapes. Depending on the features learned by your model, you very well could end up with every blue cube classified as a rod and every red rod classified as a cube. With a larger initial draw (dataset) for training, your model is more likely to be more representative of the actual data and to learn the features which completely separate objects which it is attempting to label. In our use case, we do not have an abundance of labeled ScanEagle video data from which to train our model.

Critical to training the example above are the shape labels. It is through labeled feedback,

that supervised models like CNNs learn. During training, the features of the training dataset are fed into the model; in this case they are blue, red, cube, and rod. With each training example, the model will compare the features it sees to the label its given, then adjust its internal weights to more closely match. This is perhaps the greatest restriction to just using all of the “bag” of data to train. It would take a lot of time to manually label even half of your bag of one million cubes and rods. Creating labeled datasets can be time consuming, assuming you even have access to the data for labeling to begin with. For our application, we have no labeled aerial imagery from our ScanEagles.

While we do not have access to labeled ScanEagle video data, we do have access to an abundance of unlabeled satellite data. Unfortunately, satellite data is collected with a sensor that greatly differs in quality and point of view from our low flying ScanEagle. This scenario is like using the model above to classify purple and green cubes and rods (instead of red and blue ones). While the data is similar, there is enough of a difference to potentially hinder performance of our model. There is also the problem of labeling the satellite data. Using labeled satellite data to classify roads in an area with little to no labeled aerial UAV imagery is a likely use case for the application we are evaluating. There is definite need to generate and take advantage of as much labeled data as can be provided.

With the abundance of satellite and road vector data, we have the ability to quickly generate labeled satellite images using existing software tools (ArcGIS Pro). We also have the ability to generate a reasonable amount of unlabeled ScanEagle video. Using research conducted by Ben-David et al. [1], [2] and Ganin et al. [3], we have a method which may allow use of both the labeled satellite data and unlabeled ScanEagle video to train our CNN. Ganin et al.’s [3] domain-adversarial neural network (DANN) adapts a neural network (NN) trained with labels from a source domain to an unlabeled target domain.

1.1 Problem Statement

As in Ganin et al. [3], the ability to use an existing labeled dataset (source satellite domain) to improve a model’s classification on an unlabeled target dataset (target UAS domain) is our precise use case. Until enough labeled aerial UAV video is available, there exists a need to leverage labeled satellite data. Without the ability to leverage this additional data, our goal of enabling the ScanEagle to remotely identify road networks using its on-board video

sensor is limited.

This thesis seeks to determine the performance of CNNs trained on satellite imagery when applied to the task of identifying road networks as seen from UAS using their onboard video sensors. The goal of this research is to increase the autonomy of UAS systems and their utility to the warfighter by facilitating follow on development of Artificial Intelligence (AI) techniques for threat detection, early warning, and route planning. This work will be accomplished in conjunction with the NPS's Center for Autonomous Vehicle Research using existing ScanEagle UAS. The successful development of this autonomous road detection system is a key component of an overall autonomy architecture that would eventually enable UAS platforms with the ability to independently create a more accurate picture of its operating environment. Further benefits might include the ability to operate in a Global Positioning System (GPS) denied environment and without an accurate pre-loaded map of the operating environment.

1.2 Research Questions

CNNs can provide a flexible and accurate method of classifying images and performing object detection by learning the “spatial hierarchies of patterns” (pixel values, features, and their relation to one another) that define a target object within an image [4]. NNs however are prone to over-fitting and may not perform as well outside of the environment for which they were trained. This is particularly true if they lack large datasets to help the NN learn to generalize. It is easy to find or create large datasets of labeled satellite imagery with which to train a CNN for road identification. Labeled datasets based on the ScanEagle video do not exist. This leads to a problem when trying to train a CNN that will accurately identify roads in ScanEagle imagery and presents the central question for this thesis, “Can a CNN trained to identify roads in satellite imagery be adapted to identify roads from an aerial platform?” Subsidiary questions include:

1. Can a large, labeled (road or not) dataset be efficiently created to train an accurate CNN?
2. Can a CNN be trained to identify roads from satellite imagery?
3. How does that CNN perform on a dissimilar dataset (ScanEagle video)?
4. Can that CNN be adapted to that dissimilar dataset to improve performance?

1.3 Overview

This research will be broken into five phases. The first phase includes defining the source and target domains and building the datasets that will be used to train all CNNs and DANNs. This phase will use existing satellite imagery and road vector data to extract images of roads within an operating area of interest (our test range at Camp Roberts) for the source domain. The primary challenge of building the data set this way is ensuring the use of accurately geo-rectified satellite imagery and vector data. The specific interest in the use of satellite data to train a UAS based CNN is to overcome the need for aerial training footage which is not readily available. Our target domain will use archived footage from previous ScanEagle flights in the target area. Footage from the ScanEagle was collected at 5000' mean sea level (MSL) and 3500' MSL primarily looking nadir (straight down view).

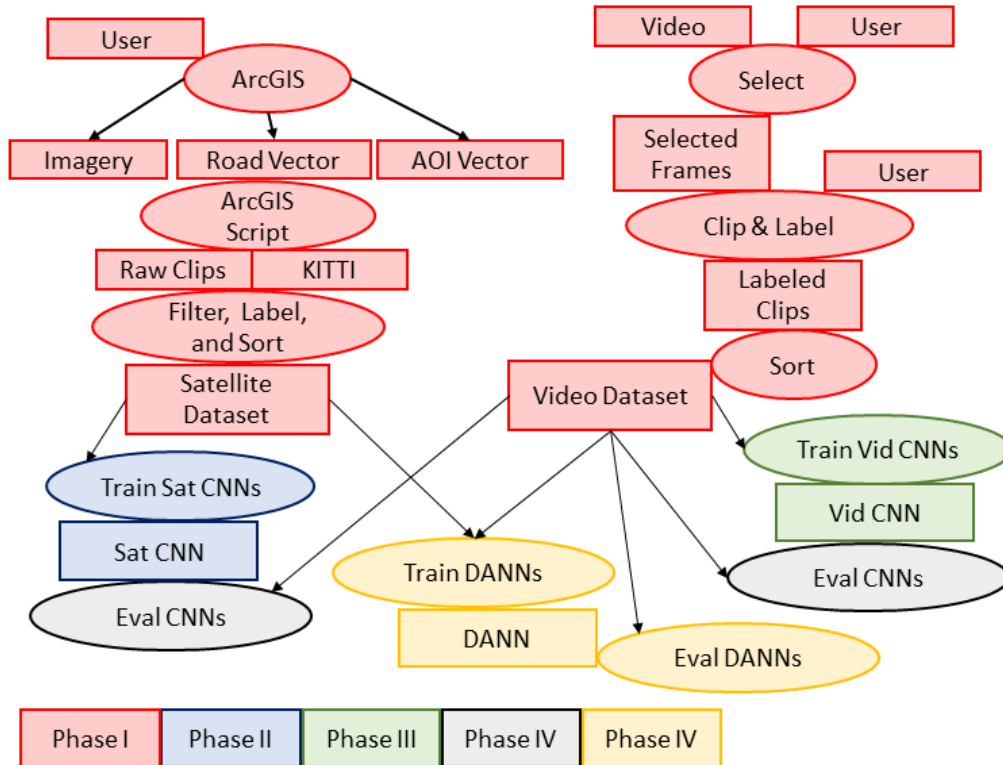
The second phase will involve training our source domain CNNs using the generated source domain dataset. We will be training several CNNs of varying type and structure on the source domain datasets created in phase one. In the third phase we will train target domain CNNs on the generated target domain dataset, also created in phase one. The fourth phase will be to evaluate our trained networks on the target domain test data. During this fourth phase, we will establish the theoretical performance limits for domain adaption (DA). The fifth phase will be to create a domain adapted CNN from our most performant CNNs to evaluate potential performance improvements provided by DANNs to this application. The overall process is illustrated in Figure 1.1 and 1.2.

In Chapter 2, we provide an overview of concepts relevant to understanding machine learning (ML), CNNs, and remote sensing from space and aerial systems. In Chapter 3, we provide an in-depth look at Domain Adaptation and a proposed solution by Ganin et al. to successfully build a cross domain classifier. Chapter 4 provides a detailed overview of the experiment data, software, hardware, procedures/phases, and evaluation criteria. Chapter 5 outlines the results of our work, while Chapter 6 covers its conclusions, limitations, and suggested future work.

1.4 Significance to the Field

AI has gained renewed interest in recent years due to the widespread availability of inexpensive, relatively powerful mobile computing platforms and access to large amounts

Figure 1.1. Thesis phases, goals, and data-flow.



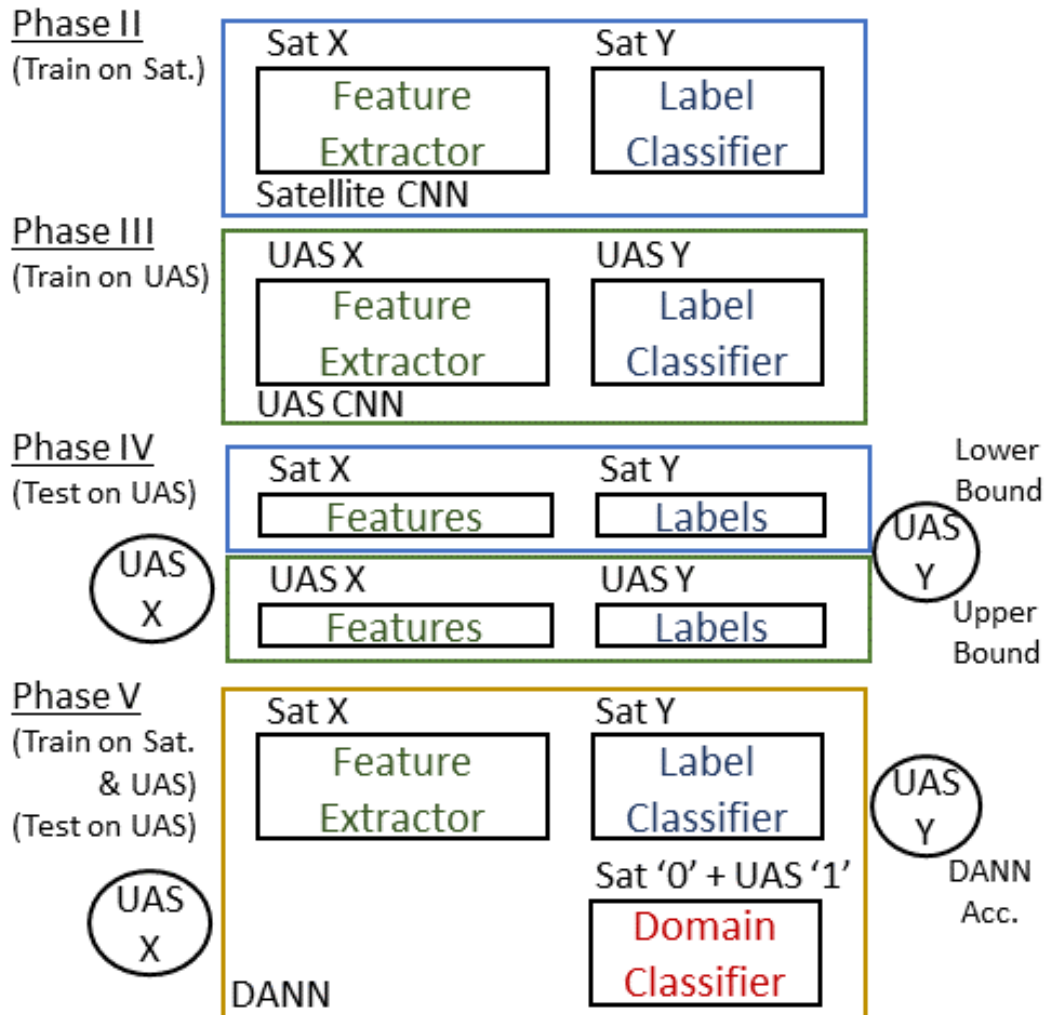
Pictured is the general data-flow and processing steps, colored by phase, for this research. We define domains and create datasets in phase I, train satellite and UAS video CNNs in phases II and III, establish lower and upper performance bounds in phase IV, and then train and evaluate DANN performance in phase V. Ovals are processes while boxes are inputs or process outputs.

of data. This renewed interest is a recognition of the dramatic implications for artificial intelligence, with some even saying that the adoption of AI will have as great, or greater, an impact as the industrial revolution was for human progress [5] — and perhaps even how we conduct warfare. Harnessing the ability to perform AI tasks on mobile platforms is part of this progression. The implications of accurate airborne computer vision could drastically increase the warfighter’s decision cycle by automating once resource intensive tasks.

This study moves the Department of Defense (DoD) towards full realization of the ScanEagle’s on-board systems. This would ultimately provide the ScanEagle with the ability to

independently create a more accurate picture of its operating environment. Further benefits might include the ability to operate in a GPS denied environment and without an accurate pre-loaded map of the operating environment.

Figure 1.2. CNN by phase.



Several CNNs will be trained in phases II and III. The X/Y “inside” each CNN represents the training and validation sets used to train them. In phase IV, the models from phases II and III will be evaluated against the X/Y “outside” each model (UAS test set) to determine lower and upper performance bounds. In phase V, DANNs are similarly trained and evaluated against the UAS test set to determine DANN performance.

CHAPTER 2: Background

2.1 Introduction

This thesis focuses at the crossroads of deep learning and remote sensing, the combination of which is a powerful mix for unmanned systems and intelligence collection. For deep learning to work in this context, a large amount of labeled data is required. Through evaluating the use of domain adaptation, we hope to circumvent this requirement and train robust CNNs regardless of our target area and the amount of labeled data available. With trained models usable for UASs as a specific goal, we seek to take advantage of the large amounts of labeled satellite data currently available.

The goal of Chapter 2 is to provide the conceptual background required to understand subsequent chapters. Here we will discuss ML (specifically deep learning) concepts: optimization and generalization, neural networks, gradient descent and back-propagation, and CNNs. We will also discuss remote sensing concepts: resolution, scale, angle, elements of interpretation, and urban considerations.

2.2 Machine Learning

For a very long time, computers have been limited to accomplishing tasks that they were explicitly instructed to do. Even when minds like Alan Turing were developing the foundational theories for AI, computers were generally unable to accomplish more than these pre-programmed tasks. Algorithms for machine learning, a subfield of AI, have been around for years, such as Rumelharts et al.’s 1986 [6] paper on NN and back-propagation, yet these methods were still out of reach for most computers. It was not until they had access to large amounts data and inexpensive computation that they began to take off [7]. Through ML, computers now have the capability to “adapt to new circumstances and to detect and extrapolate patterns” [8]. We must merely provide the raw input and the computer figures out the rules required to provide new answers without human input [4]. This newfound power for computers makes ML a critical area for study, as they have the ability to learn

new connections and patterns and perform on a level that can surpass humans.

2.2.1 Functions, Hypotheses, and Error

Binary functions, in the context of ML, are the true mapping of data inputs to their corresponding binary labels (is/is not, true/false, spam/not spam, cat/dog, etc.). Consider mapping of predicates to their truth values, $(1 + 1 = 2) \rightarrow \text{True}$ or $(1 + 5 = 3) \rightarrow \text{False}$, this true mapping is the domain's function, $f : X \rightarrow [0, 1]$. In a simple application, such as this, the function of a domain is known. In cases where this function is not known, ML will use sample data to create an estimate of this function. This estimate is a function or model called a hypothesis, $h : X \rightarrow \{0, 1\}$ [2].

These functions can be any of a range of mappings from one domain to another. In ML, they are hypotheses that range in the form from a simple logistic regression formula to an incredibly complex CNN. The range of these hypotheses that our learning algorithm can consider as a solution to the true mapping is called a hypothesis space, later referenced as $\mathcal{H}$. Adjusting our hypothesis space can limit the range of possible solutions and complexity of a problem, as is done by Ben-David et al. when they find the optimal $h \in \mathcal{H}$ [2]. We limit our $\mathcal{H}$ to CNN with a binary classifier.

The “risk,” or error, of a hypothesis is the probability that that hypothesis diverges from the true labeling function. The actual error between a hypothesis and function over the entire source domain, $\mathcal{D}_S$, is defined (with shorthand notation) as

$$\epsilon_S(h, f) = \epsilon_S(h) = E_{X \sim \mathcal{D}_S}[|h(x) - f(x)|] \quad (2.1)$$

The empirical error, $\hat{\epsilon}_S(h)$, is the error between a function and its hypothesis actually measured for a given sample of $\mathcal{D}_S$ [2]. The important thing to remember is that knowing the actual error, ϵ_S , requires use of an “oracle” which knows the true distribution of h and f [9]. In many cases, this is not possible. The error function presented above, known as the L_1 distance, is the error function used as a measure of domain divergence by Ben-David et al. [2] (to be discussed further in Chapter 3); other error functions exist.

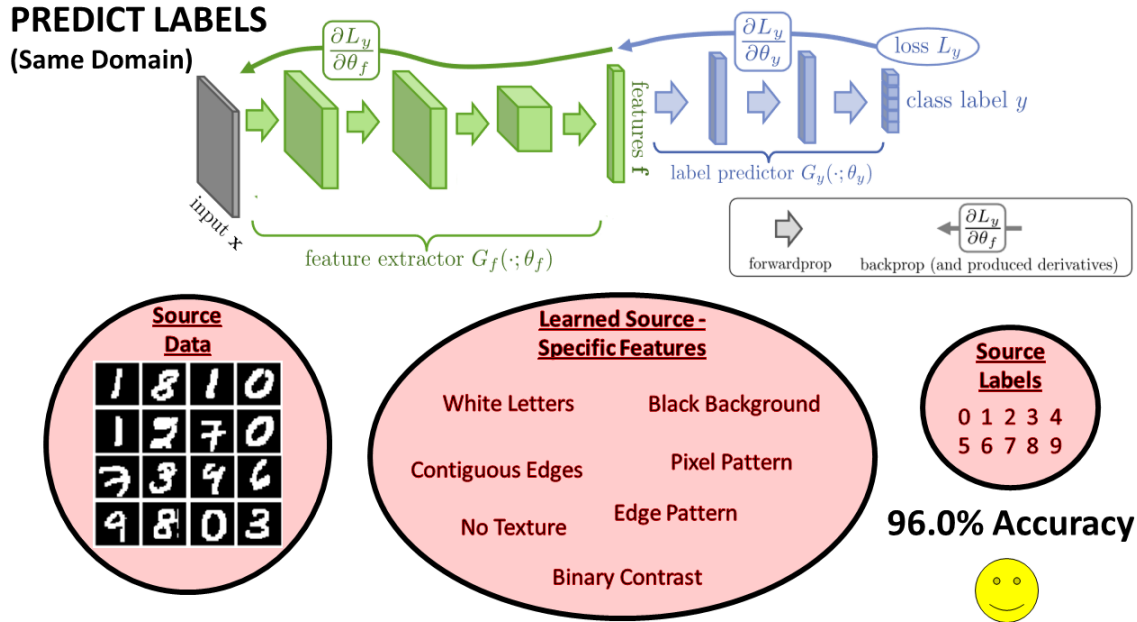
2.2.2 Optimization vs. Generalization

When training many ML models, we are actually working to find the optimum of a specific model criterion with respect to our training data. This process, called optimization, moves a given $f(x)$ towards either a desired minimum or maximum by changing x , the function or model weights. This $f(x)$, is referred to as the error function, as defined previously, but is also commonly referred to as the cost function, objective function, or loss function [9]. These terms are generally synonymous. The desired end-state of optimization is either a global minimum or global maximum. These points are either the absolute lowest or highest value for $f(x)$, respectively. Optimization, like the error function, can be computed in many different ways. Typically, it is accomplished through an algorithm called gradient descent (explained in the next section) or some modification of gradient descent. Some of the more well known forms of gradient descent are stochastic gradient descent (SGD), SGD with Momentum, Nesterov Momentum, Adagrad, RMS Prop, and Adam. These are all explained more thoroughly in [9] or [10].

Generalization is a term that refers to how well a model performs on data it has not previously seen [9] (see Figures 2.1 and 2.2 for illustration). Minimizing generalization error (error on a new input) is precisely the goal of ML. We optimize a hypothesis, or model, through training with the specific intent of increasing its ability to generalize to new data. Further increased generalization is the goal of domain adaptation and our research. We seek to not only create a model that generalizes well on data similar to what it was trained on, but to also force a model to generalize to a whole new domain without access to the data labels required to optimize that model for that domain.

An important consideration for any ML specialist is the match between the capacity of the chosen $\mathcal{H}$, the complexity of the learning task, as well as the volume, type, and quality of data available for training [9]. The performance of a model on its training data diverges from its generalization error as the model capacity increases or the amount of training data decreases. The quantity of training data is quite easy to figure out; the concept and measure of capacity is not. Several theories exist to provide a measure for model capacity. The measure used by Ben David et al. to bound domain divergence is called the Vapnik-Chervonenkis (VC) dimension (VC dimension, d). The VC dimension is specific to binary classifiers and “is defined as being the largest possible value of m for which there exists a training set of m different x points that the classifier can label arbitrarily” [9].

Figure 2.1. Typical, non-domain adapted, feed-forward CNN. Adapted from [3].

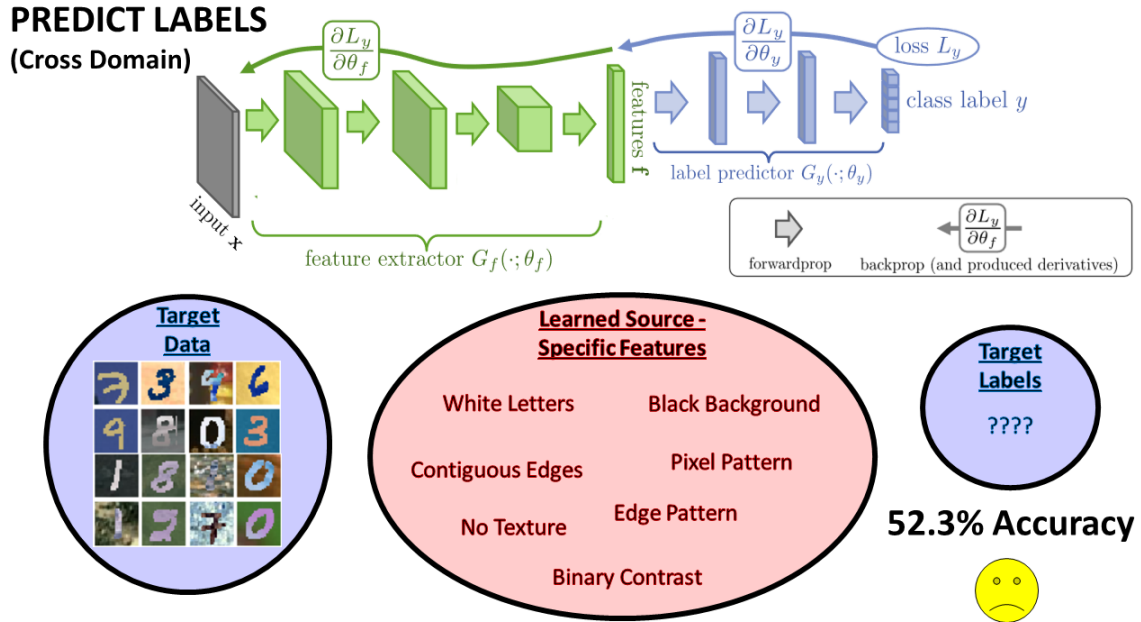


Illustrated is a hypothetical, non-domain adapted CNN trained to predict labels in the same domain it was trained on (MNIST data in this case). The accuracy shown is from [3].

2.2.3 Neural Networks and Deep Learning

Many ML algorithms consist either of a single transformation (linear regression), a transformation then classification (logistic regression), or even a series of decisions (trees). Neural networks are similar in their simplicity; each layer consists roughly of some transformation and subsequent activation. The “network” part of “neural network” is due to the composition of many “units” in layers (common transformations) connected to one another into a series of chained functions [9]. This combination of layers (functions) is intended as a way to replicate, in a non-linear way (due to non-linear activations), the true mapping of an input to a specific category or numerical output [9]. While a threshold is not officially established, once a chain of functions reaches a certain length it is considered to be a “deep learning” model. The term deep learning is derived from this [9], but means much more. Deep learning is the automated exploration of data and the creation of “new” higher and higher levels of feature abstractions from initial raw inputs.

Figure 2.2. Typical, non-domain adapted, feed-forward CNN. Adapted from [3].

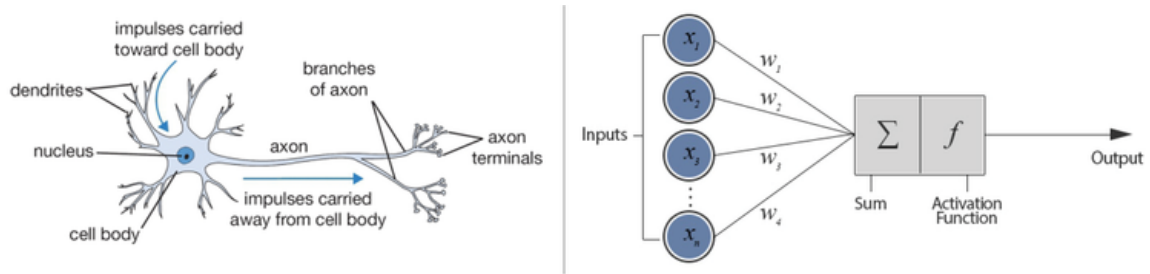


Shown is the same hypothetical, non-domain adapted CNN from Figure 2.1 predicting labels for data from a domain different from that which it was trained on (MNIST-M data in this case). The model does not generalize well and performance suffers. The accuracy shown is from [3].

The “neural” part of “neural network” is inspired by mathematical modeling of how actual neurons work. This is explained especially well in [11]. Each unit of a neural network is equivalent to a neuron in the brain. A neuron receives an input from another neuron at a synapse. Each synapse transfers this input through a dendrite (with a transformation) to the cell body. The cell body evaluates a sum of these dendritic inputs and activates if a certain threshold is met. This activation is transmitted through the cell’s axon to other interested neurons (see Figure 2.3).

Typical transformations within units of a neural network are called affine transformations. They are linear functions that consist of a weight, w , and bias, b : $\hat{y} = w^T x + b$ [9]. Affine transformations are then followed by a non-linear activation (otherwise the combination of two linear functions is still a linear function). Activation functions that operate on the results of this transformation are varied. They include functions such as the Rectified Linear

Figure 2.3. An artificial neuron compared to an actual neuron. Source: [12].



A comparison between an actual neuron and an artificial neuron. Note for both, the inputs are “weighted,” “learned,” and the results output through an “activation” function.

Unit (ReLU), leaky ReLU, Maxout, Sigmoid, Tanh, etc. [9], [11] and can also include functions such as a dropout layer, or batch- or layer-normalization layer. While these optimizations are included in the CNNs designed for this paper, their detailed explanation is beyond the scope of this thesis. Generally though, dropout is a method of regularization that prevents overfitting and promotes learning of new features while batch-/layer- normalization are intended to prevent saturation of activation functions and promotes faster learning of features.

A great analogy for understanding neural networks, CNNs, and steadily increasing levels of abstraction is explained in Tyler Renelle’s “Machine Learning Guide” podcast [13]: Assume you are observing a company (NN) that was created to classify images as either a face or not a face. This company is broken up into departments (layers), filled with specialists (units), whose sole purpose is to take in inputs and decide if they belong to a certain “feature” of a face. These departments are arranged in a hierarchical fashion (one feeds “features” to the next). Initial departments take in groups of pixels only (a convolution applied to a window over an image). These lower departments make decisions about whether these pixels form patterns such as a horizontal edge, vertical edge, or diagonal edge and its orientation (left to right or right to left). Each unit in this department is tasked with one, and only one, type of pattern. If their assigned pattern exists within their specific window, they provide a “yes” (activation) to the next department up on the company hierarchy. This next department does the same thing as the first department, except instead of looking at individual pixels, “units” are looking at the list of “yeses” of the previous department. The “units” in this

second department decide on whether or not they are receiving indications for their own set of patterns, such as vertical or horizontal lines. Each department up the hierarchy begins to operate at a higher and higher level of complexity. Lines become groups of lines. Groups of lines become a nose, eyes, and mouth. Groups of a nose, eyes, and mouth become a face. At some point these high level features are provided to the boss at the top of the company (classifier). The boss, when training his company, heads over to his evaluator (loss function) to compare his company's results (face/no face) to the actual label of the input that his input department received. He takes the distance (loss) between the true answer and his produced result back to the department that fed him his high level features. They adjust the calculations they use to prompt their "yes" output, then pass their feedback (gradient) on to the next lower department. This feedback passes all the way to the initial input department that operated on pixels alone. This process continues until all training data has been run through.

As seen in this analogy, NN are essentially chains of transformations and "decisions." Ganin et al. describes the label prediction component of a DANN in this way: as a chain of functions, $\mathcal{L}_y(G_y(G_f(x_i; W, b); V, c), y_i)$, where the output of one function is input into the next: $G_f(x_i) = z \rightarrow G_y(z) = s \rightarrow \mathcal{L}_y(s) = loss$. This is the quintessential feedforward network: a function feeding a function feeding a function, etc. This arrangement can be illustrated using a computational graph. For feedforward networks, these graphs are acyclic with nodes representing variables and edges representing some function or composition of functions (with one output value) [9]. The layers of NN that fall between the input layer and the final layer, which provides the desired output, are called hidden layers. Hidden layers are the equivalent of the inward facing departments of the company analogy. They do not produce the final decision or value nor do they interface with the data directly.

2.2.4 Gradient Descent and Back-Propagation

The consistency in concepts across various ML algorithms is convenient. Gradient descent remains the same for neural networks as it is for shallow machine learning algorithms like linear or logistic regression; we calculate the gradient of the cost function with respect to the input. The only difference is the inclusion of gradients for non-linear activation functions (like ReLU) as well as a process called back-propagation. Understanding these two concepts, gradient descent and back-propagation, is critical to understanding the theory

and practice proposed by [2], [3]. Their theory, on how to modify this process, is the basis for this thesis and is explained in more detail in Chapter 3.

Understanding gradient descent (amazingly first discussed by Louis Cauchy in 1847) requires a basic calculus refresher. A derivative is a function which defines the instantaneous slope of given function with respect to some value [14]. A gradient is similar to a derivative. It is a vector of derivatives of a function taken with respect to each component (partial derivative) at a specific point [14]. In gradient descent, the function we find the gradient of is our cost function. This gradient is taken with respect to the weights of our model. In the case of a NN, these are the weights associated with each unit of the NN. Taken as a whole, this cost function, with each weight a dimension, we have a multi-dimensional hyperplane with a global cost minima that we seek to find to minimize error. In deep learning, we use gradient descent to find that minima. By finding the gradient vector, we essentially find the direction to move our weights and then adjust them accordingly. The generic gradient descent formula is [9]:

$$\mathbf{x}' = \mathbf{x} - \epsilon \nabla_{\mathbf{x}} f(\mathbf{x}) \quad (2.2)$$

In this equation, $\mathbf{x}$ is the vector of weights. The learning rate, ϵ , is a hyperparameter used to control the influence of each gradient “step” (a hyperparameter is an unlearned value set by the user to adjust ML algorithm behavior) [9]. The subtracted value is what drives the descent through each iteration of the algorithm. This portion of the equation is what is typically modified by the optimization schemes mentioned in the last section.

Analogies, such as the one discussed in Stanford’s CS231n [15], provide a great visual for understanding the gradient descent algorithm. Here they illustrate the hyperplane, that is the cost function, as a mountain range full of peaks and valleys. $\mathbf{x}$, is the set of weights that determine our location on the hyperplane and the elevation is the cost associated with this position. Our goal is to minimize our elevation (cost, $f(\mathbf{x})$) by descending the hill we initialized our weights to. We do this by finding the direction of the uphill slope (gradient, $\nabla_{\mathbf{x}} f(\mathbf{x})$) at our current position. Our learning rate, ϵ , then determines the size of the step we take in the opposite direction (“-”). We continue this process from our new position, $\mathbf{x}'$, until we reach some local or global minimum.

In shallow learning algorithms, finding the gradient of the cost function is relatively straightforward as we have only one layer of weights to find the partial derivatives for. Values feed forward through the algorithm to the cost function and the gradient is computed directly for each weight. In NN, however, we have layer upon layer of weights to compute the gradient for. During forward propagation, inputs work their way through each layer of the network until it reaches its last layer and finally the cost function, as depicted by its computational graph. Our gradient must then be calculated, moving back towards the input, for each layer of the network. This backwards transmission of the gradient is called back-propagation. It is fundamentally the chain-rule of calculus [9]. An example of the chain rule is seen in equation 2.3. Using the chain rule allows us to essentially consider each unit of the network locally, with inputs from the left and upstream gradients from the right. It is during the back-propagation of the gradient that Ganin et al. propose their changes to make DANNs work [3].

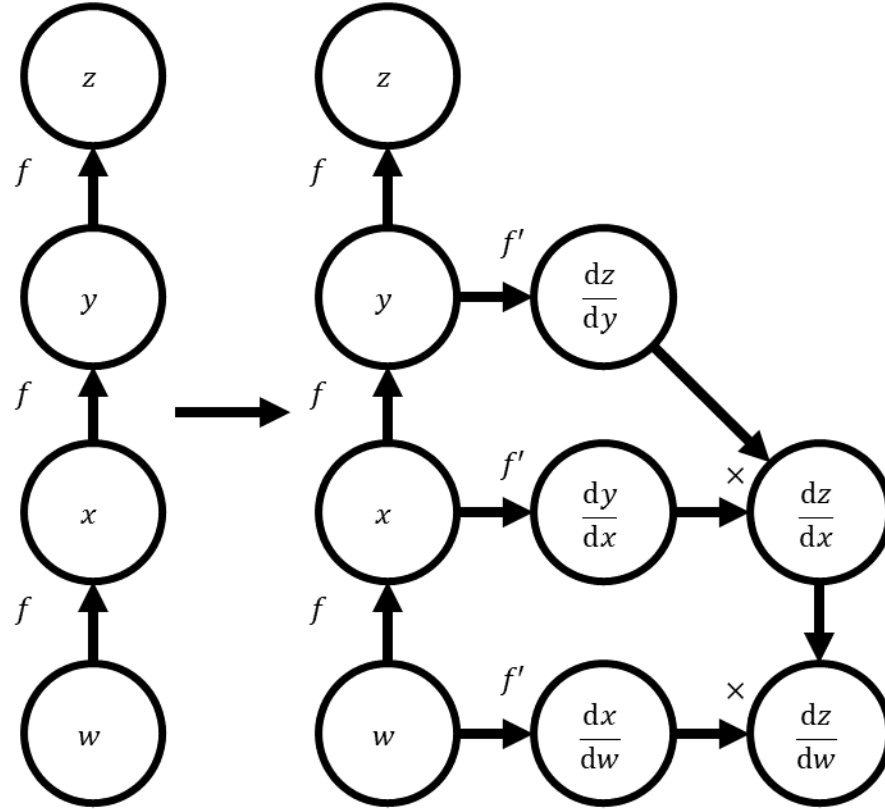
An example of the chain rule is shown in equation 2.3 and illustrated in Figure 2.4:

$$\begin{aligned}
& \frac{\partial z}{\partial w} \\
&= \frac{\partial z}{\partial y} \quad \frac{\partial y}{\partial x} \quad \frac{\partial x}{\partial w} \\
&= f'(y) \quad f'(x) \quad f'(w) \\
&= f'(f(f(w))) \quad f'(f(w)) \quad f'(w)
\end{aligned} \tag{2.3}$$

2.2.5 CNNs

CNNs have been around since at least 1989 when discussed by Yann LeCun in his “Generalization and network design strategies” paper [16]. They are a form of NN that perform a transformation of their input using a convolution: not much more complicated than that. These convolutions are a linear operation on an input that takes into account the topology, or “spatial relationships,” within that input. These topology aware operations work well on time-series data, single channel images (greyscale), multi-channel images, multi-channel images over time (color video), etc. Such input, I , is transformed through convolution by the kernel, K , to produce a feature map, S [9]. For a two dimensional space, this would look like [9]:

Figure 2.4. A computational graph demonstrating the chain rule. Adapted from [9].



$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(i - m, j - n) K(m, n) \quad (2.4)$$

For CNNs purpose built for processing images, each convolutional layer will process an input with height, width, and channel depth. They will have a set of kernels, also called filters, containing the weights to be trained. These kernels will have a spatial extent denoting the size of the input to convolve. Each layer will also have a hyperparameter called a stride, which denote how much the kernel “window” will slide across the input with each convolution. In many cases, a padding size will be set to ensure the combined stride and

kernel size match the input and do not result in invalid values [17].

2.3 Remote Sensing

“Remote sensing” is a term coined by the Office of Naval Research in the 1960s [18]. In short, it truly means “sensing remotely.” In more specific terms:

Remote sensing is the non-contact recording of information from the ultraviolet, visible, Infrared (IR), and microwave regions of the electromagnetic spectrum by means of instruments such as cameras, scanners, lasers, linear arrays, and/or area arrays located on platforms such as aircraft or spacecraft, and the analysis of acquired information by means of visual and digital image processing [18].

One substantial benefit of remote sensing is its ability to provide monitoring of entire regions with just one pass of a remote overhead sensor. An example of this is demonstrated in a 2008 paper by Hestir, et al. [19]. Using remote sensing methods, they show how invasive wetland and aquatic plants can be monitored across an entire region (in this case the full Sacramento-San Joaquin River Delta), based on the spectral characteristics of each species. This method provides a land resource manager a highly accurate map of species spread and movement year-to-year without the time consuming and costly need to travel every channel, bay, and estuary to find, identify, and map the location of those species.

2.3.1 Resolution

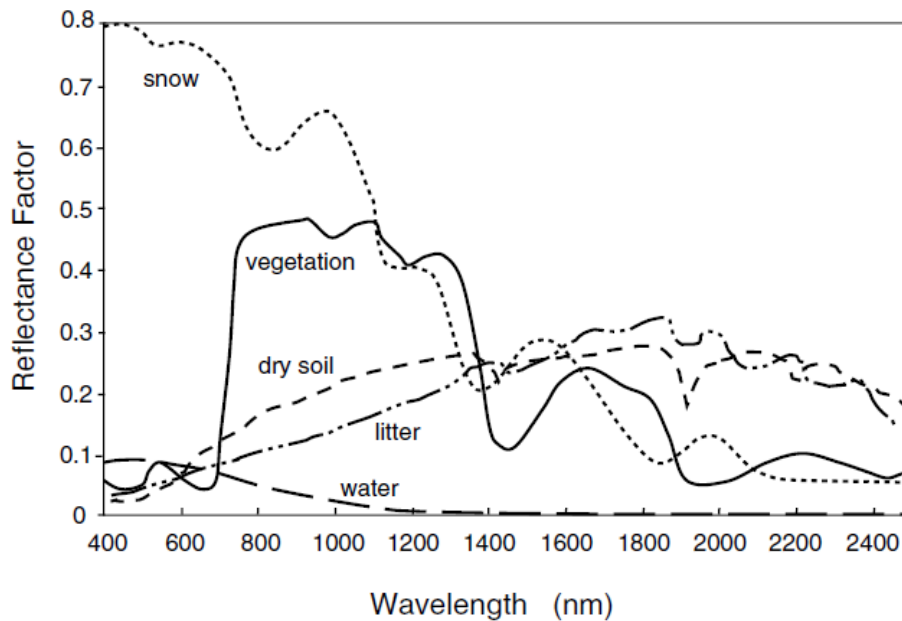
Remote sensing provides clear contrast to in situ forms of data collection (such as a researcher in the field with a GPS and portable spectrometer or a small mobile team on patrol). Passive remote sensing occurs without the target ever knowing it was being observed through collection of electromagnetic energy emitted or reflected by the target. This passive collection is characterized through three forms of resolution: spectral, spatial, and temporal.

The range of wavelength intervals in remotely sensed data, both their number and width, is referred to as the spectral resolution of a remote sensing system. Remote sensing systems that collect a range of electromagnetic bands are called multispectral. Systems are called hyperspectral, when “multispectral” refers to hundreds of bands [18]. Each collected band

is equivalent to a channel and provides a sampling of different parts of the electromagnetic spectrum. For example, a “red channel” or band in a typical red, green, blue (RGB) image displays information collected in the red portion of the electromagnetic spectrum. Objects are, of course, reflective in more than just the red band, as is seen in Figure 2.5.

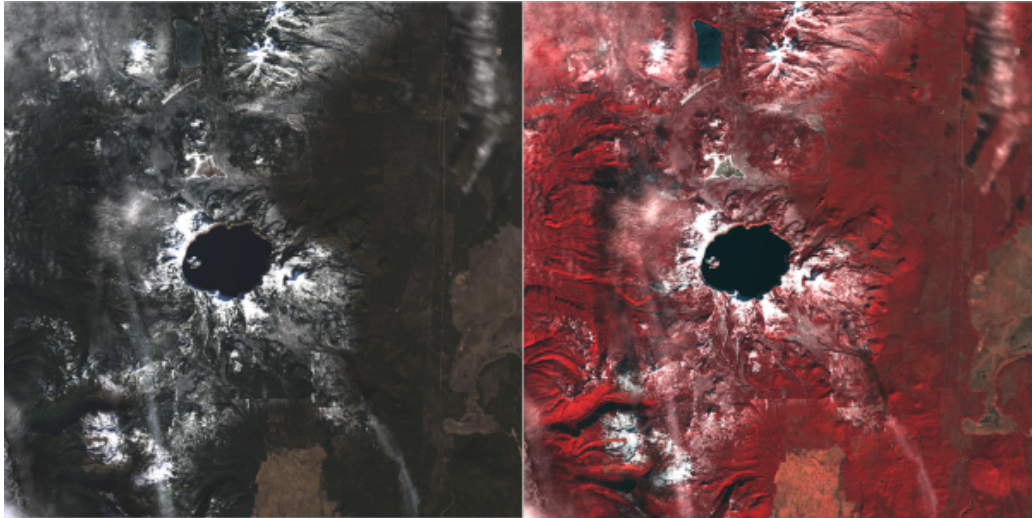
Since our eyes (and computer screens) are designed for RGB, these multispectral or hyperspectral images are often displayed as some composite of desired bands. For instance, “false color IR” images are a three channel image where “red” is representing near-IR values, green represents red, and blue represents green, see Figure 2.6. A blue car in reality would be appear dark in the image, since its blue color is not represented. A green tree would appear bright red, since plants reflect substantially more near-IR light than they do green light. A green tank hiding amongst those green trees would be displayed as a blue tank hiding amongst red trees.

Figure 2.5. Example spectral reflectance curves for different materials.
Source: [20].



Shown are sample spectral reflectance curves for various materials. The higher the spectral resolution of an image, the higher the amount of spectral information captured by that image. Notice the small spectral resolution of typical panchromatic imagery (all visible bands recorded as one channel) which captures a single band from 450-800 nm.

Figure 2.6. Natural and false color infrared composite satellite imagery. Adapted from [21].



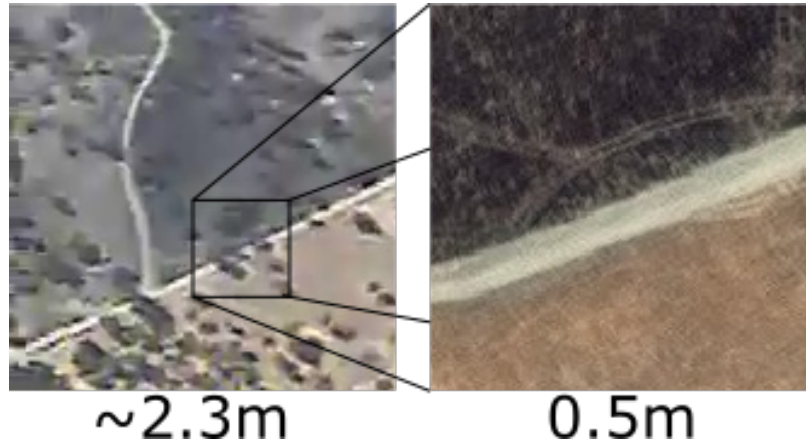
On the left is a natural color composite of Crater Lake, Oregon. On the right is a False Color Infrared composite of the same area. Notice how different features are more apparent with each combination of bands. The Humboldt State University website includes a great web page for exploring different composite images (see citation for figure).

In addition to spectral resolution, a system will have a spatial resolution. This spatial resolution is an indication of how small an object can be discerned. It is usually denoted as the area represented by each pixel in the image [18]. For instance, a 0.25 meter spatial resolution means that each pixel represents a 25 by 25 centimeter area on the ground. This level of system resolution could be expected to create an image where the computer screen used for writing this thesis would appear as one or two black pixels after being thrown out of the window. The rule of thumb for determining how much spectral resolution is needed is to half the smallest feature you need to discern by its smallest dimension [18]. If the spatial resolution were the same size as the object, the object could be split across two pixels and not be detectable due to mixing of the spectrum detected by the portion of the sensor recording that area on the ground. Halving the smallest dimension ensures the object will completely fill at least one pixel (though from a second story window, this monitor might fill several pixels). This value can be calculated as:

$$D = \beta H \quad (2.5)$$

where D is the diameter of the circular area sensed by an individual cell/unit of a sensor (like an individual CMOS photodiode), β is the instantaneous field of view (IFOV), and H is the height of the sensor in meters Above Ground Level (AGL). IFOV differs from field of view (FOV) in that FOV is the circular area detected by the entire system as opposed to an individual sensor in that system. Different sensor lenses provide different sensor FOVs: Narrow $< 60^\circ$, normal $60 - 75^\circ$, wide-angle $75 - 100^\circ$, and super-wide-angle $> 100^\circ$. Increasing altitude or the FOV of the sensor increases how much ground is captured by a single unit of the sensor (turned into a single pixel). An example of different spatial resolutions can be seen in Figure 2.7.

Figure 2.7. An example of different GSDs.



These are two scenes of approximately the same area within each dataset used for this thesis. Both images are 160x160 pixels in size with different GSD (as displayed below each image). The left clip is extracted from the ScanEagle video and shows an area approximately 370x370 meters. The right clip is extracted from the satellite imagery and shows a sub-region of the left clip of 80x80 meters. Note the amount of detail (information) absent in the lower-resolution ScanEagle image.

In addition to spectral and spatial resolution, remote sensing systems also have temporal resolution. Temporal resolution refers to the frequency that the sensor revisits and records a specific area [18]. For instance, if an aerial remote sensing platform collected imagery before the computer monitor was thrown out the window, then collected imagery of the

same back yard one week later, it would have a temporal resolution of seven days. All three forms of resolution are important to the analysis of the collected data as they together provide important relational information to an analyst (or computer).

Remotely sensed data also collects location information (ex. latitude, longitude, and elevation). With this information as the basis for each pixel, an analyst could then determine what was at a location (spatial/spectral resolution) and where it was moving to or how fast it was changing. For instance, consider an analyst evaluating two false color IR images alongside two regular images of the same war zone taken a week apart with 0.3 meter resolution. They detected a group of square blue objects, sized 12 pixels by 26 pixels, tucked into a red treeline in their false color IR image. This same group of blue objects is not visible in their normal color image. In the second set of images, this group of blue objects appeared just over 3300 pixels away and were clearly visible in the color images (they used the computer to measure this). The analyst could conclude that a group of tanks (about 3.6m x 7.8m each) hidden in the treeline had moved a kilometer away and were no longer trying to hide their location.

2.3.2 Scale

Scale is an important characteristic of collected remote sensing data and is required for determining many of the imagery features outlined later in 2.3.4. It provides a relative measure of size for objects or features in an image [18]. Scale can be determined in two ways: (i) comparison of objects in the data to objects in real life or (ii) using information from the collection platform at the time of collection [18]. When determining scale from the data itself, scale is calculated as [18]:

$$s = \frac{ab}{AB} \quad (2.6)$$

where ab is the size of the object in the data and AB is the size of the object in real life. The second way to determine measure is based on the sensor focal length f and its height, H , AGL [18]:

$$s = \frac{f}{H}. \quad (2.7)$$

From this second equation, one can see how scale might vary, even, within a single image. Any sensor tilt during collection, or varying elevation of the ground below, can cause variation of H , thus scale, within a single image. H can even vary from the center of an image to its edge. This is why the average scale for an image is generally what is presented [18].

2.3.3 Collection Angle

In remote sensing, nadir is the reference point for evaluating the collection angle of a sensor. It is point on the ground directly below the sensor [18]. If a photograph is taken straight down from an aircraft, that photograph was taken looking nadir. If a photograph was taken looking towards the horizon from that same aircraft, the photograph is off-nadir. Understanding the angle in which an image is captured provides a tremendous amount of context for its analysis and the imagery's limitations. Off-axis imagery can eliminate the need for a collection platform to fly directly over its target area and can even collect more imagery of the ground from this vantage. If done correctly, two off-nadir views of the same terrain can even provide what is called stereoscopic parallax [18], which introduces opportunities for measuring the height of image features.

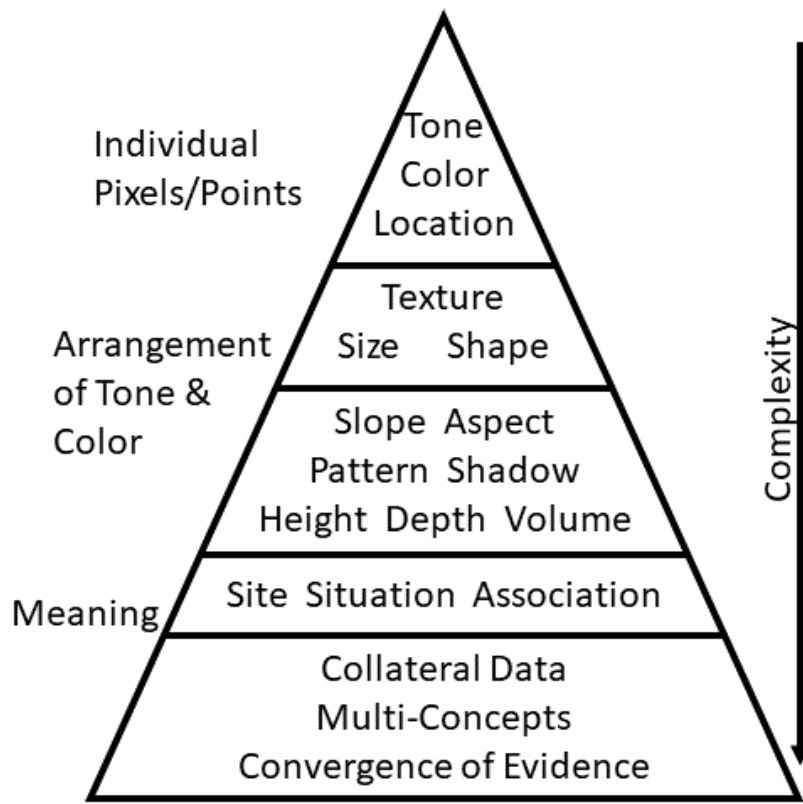
For classification purposes, vertical images are those collected where the difference between the sensors axis and a line perpendicular to the ground, nadir to the sensor, is less than 3° . Images are considered oblique when they are not vertical. Low-oblique images are oblique images that do not contain the horizon. Images that do contain the horizon are called high-oblique [18].

2.3.4 Imagery Features

Remote Sensing Analysts rely on "Elements of Image Interpretation" to glean information from the data they have collected (see Figure 2.8. Our trained CNN will learn many of these features and can even be designed to incorporate these elements. These elements include "location, tone and color, size, shape, texture, pattern, shadow, height and depth, volume, slope, aspect, site, situation, and association" [18]. These elements appear in somewhat of a hierarchy, where location, tone, and color form the basis for size, shape, and texture. These elements gradually increase in complexity and abstraction, each depending on the more simple elements below it [18]. This roughly parallels the process of feature extraction

done by a deep CNN. One might argue however, that the results provided by a CNN are far more objective and repeatable than that of the analyst who might be operating on a sleep deficit from working on his masters' thesis.

Figure 2.8. Elements of Image Interpretation. Adapted from [18].



The hierarchy of the “Elements of Image Interpretation” used by analysts to construct meaning from an image. Adapted from [18].

Location

Each pixel in a rectified remote sensing image correlates to a specific x, y coordinate of a map projection (ex. Universal Transverse Mercator [UTM], etc.). This is done during post-processing of the raw data and involves using the collection platform's GPS location at the time of collection or using known GPS ground reference points. Rectifying an image is the process of stretching/compressing it to fit known coordinates and elevations on the ground and/or the attitude of the sensor [18].

As with CNNs, an image analyst can infer that pixels near each other are related spatially. This relation is fundamental to all other elements.

Tone and Color

Tone is the term in remote sensing that refers to the shade of gray, on a black to white scale, that a specific band or channel of electromagnetic energy appears [18]. This is equivalent to saturation level in a hue, saturation, value (HSV) representation of color. It would also be equivalent to the value of a single channel in a gray-scale image or the value of, for example, the red channel which might be representing near-infrared electromagnetic energy. Not only does tone help the analyst identify higher level elements of interpretation, it also adds a limitation to what can be interpreted. In remote sensing, it is understood that analysts can tell apart “only about nine shades of gray when interpreting continuous-tone, black-and-white photography” [18]. This results in lost information if there are actually 256 shades of gray available in the image. Computers do not have this limitation.

The pattern in which “colors” appear can help with identifying the type of object in an image. A pixel in a field of green grass would have lower values for blue and red than it would for green. If we extend this concept across the electromagnetic spectrum to other bands of energy in our multi- or hyper-spectral imagery, we see that materials create a spectral reflectance curve that acts, essentially, as a spectral signature for that material [18] (see Figure 2.5). All green plants, for instance, share that green grass pattern of low red/blue, higher green, with a major increase in near-IR (since plants reflect near-IR energy). Different types of color composites (a specific match of bands to RGB image channels) provide different advantages to the analyst, as illustrated in the tank example from Section 2.3.1. CNNs can interpret every band simultaneously.

Size and Shape

An object’s size provides one of the quickest ways of identifying what an object could or could not be. A semi-truck is larger than a car, a major freeway is wider than a dirt road, and a football field is larger than a tennis court. While relative size aids in identifying objects, the actual scaled size of an object is helpful as well. If the scale of an image is known, and it has been rectified, the actual dimensions of objects in an image can be measured to help

identify it. For instance, rails are spaced 4.71ft apart in the United States and tractor trailer rigs are generally 45 to 50 ft long [18].

An object's shape is also helpful in its identification. An object with clear lines and sharp corners is likely man-made (like the Pentagon in Washington, D.C.), while an amorphous object is likely natural (like a tree) [18]. Roads are long and continuous. Buildings typically have sharp corners. Rivers are long and continuous, but branch, merge, and meander. Football fields are rectangular while baseball fields look like a fan.

Texture and Pattern

"Texture is the characteristic placement and arrangement of repetitions of tone or color" and is typically described as smooth (uniform), intermediate, or rough (coarse) [18]. A jungle would appear coarse as compared to an airport runway, which is necessarily quite smooth. This apparent texture is dependent on the scale of the imagery [18]. A body of water might appear smooth at low scale, but coarse at high scale when you can see individual waves and ripples across its surface.

"Pattern is the spatial arrangement of objects in the landscape" [18]. It is also helpful in discriminating objects. For instance, Capitol Region freeways and parking lots might differ only in their gridded pattern of tightly packed cars. An orchard would appear as a grid of trees while a plowed field would have repetitive rows of plants.

Shadow

Imagery is ideally collected around solar noon to avoid shadows and observation of objects by those shadows [18]. While shadows might hide data, they provide plenty of additional information on their own. Shadows can reveal the relative heights of objects or even the identity of the object itself. The Washington Monument might cast a telling silhouette for instance when viewed from nadir it is unidentifiable [18].

Height, Depth, Volume, Slope, and Aspect

These data features are fairly high level and often require photogrammetry and/or stereoscopic instruments during collection [18]. Much can still be determined about these features even without that higher level analysis. Due to "relief displacement," cues about height

or depth (and thus volume) are still present [18]. Unless the Washington Monument was directly nadir to the collecting sensor, you should be able to see it sides. Relative to other buildings on the National Mall, there is a lot of “side” to the National Monument. This is a clear indication that it is substantially taller than surrounding buildings.

Site, Situation, and Association

Site, Situation, and Association are the more interpretive features of remotes sensing data. They are often features are known about the imagery before even looking at it. “Site” refers to the imagery location’s physical and socioeconomic characteristics [18]. This is similar to the concept of prior probability. For instance, one would know that if imagery was collected from an area with lots of farming that you might expect a plowed field to be a farm. Or if the imagery was collected from a city, a uniform green field surrounded by trees is a park and not a meadow. If the imagery was from a the Mohave Desert, all the light brown “stuff” is probably sand or gravel.

The situation of objects in imagery is the relative placement and orientation of objects [18]. A suburban neighborhood will contain lots of houses all facing the street. The association of objects is the likelihood of finding certain objects together [18]. This is similar to the concept of posterior probability. For instance, we can expect to find a school mixed amongst the houses of a suburban neighborhood. We could expect to find planes if we found an airport, or cars if we found a road. Powerlines would likely be present near a power plant.

2.3.5 Urban and Suburban Considerations

Urban and suburban environments introduce a wide range of unique material and shapes for the remote sensing analyst. They also introduce unique challenges compared to other remote sensing targets. Features of interest in an urban/suburban settings are often smaller or mixed in close proximity to other objects. A pixel in an urban setting might contain spectral information for asphalt, vegetation, and metal all at once). Working around this requires techniques for “spectral mixture analysis” or the use of “multi-sensor data fusion” [18]. Sometimes materials are reused for different purposes, such as asphalt on a road or asphalt on a roof [18].

Challenges such as these require higher spatial resolution; less than 5m is best. This

higher spatial resolution allows the analyst to rely on more of the “geometric” elements of image interpretation (shadow, pattern, orientation, texture, shape, and size) [18]. For the urban/suburban environment, spatial resolution is far more important than spectral resolution. At a minimum however, there should be sufficient contrast between objects of interest and their expected background. For instance, roads in the desert surrounding Yuma, Arizona are made of the same dirt as the exposed, un-vegetated soil surrounding them. They are the same color and at times are indistinguishable from the surrounding desert. Enough spectral resolution to provide contrast between these roads and the surrounding area is essential.

Roads

The minimum recommendations for a detailed analysis of road infrastructure is much more stringent than a general urban/suburban setting. [18] recommends a minimum of 25cm to 50cm of resolution for measuring road width and type. This task is best with visible color or panchromatic (all visible bands recorded as one channel) imagery. If only an approximate road centerline is required, spatial resolution may be as high as 30m.

2.4 Summary

While the parallels are rough, there seems to be a substantial overlap in how a CNN might learn to interpret a remotely sensed scene and how a remote sensing analyst might interpret a scene. Both rely heavily on a hierarchy of abstraction based on spatial, spectral, and temporal aspects of remotely sensed data. In a CNN, a convolution can be multi-dimensional and could very well apply across these three aspects: just as a remote sensing analysts would consider the three. The lowest levels of abstraction are entirely the same; pixel location, color, and tone create the base features. Combinations of these base features are joined further and further into more abstract concepts. The basic difference is that the image analyst can tell you their thought process (based on their subjectivity), while abstract features of a CNN can remain unexplained (though are repeatable and objective).

The difference between a CNN and an imagery analyst, as applicable to this thesis, is that the imagery analyst can apply knowledge learned previously on other imagery, to help them classify objects in new areas, new types of imagery, and different forms of color composites. If, as an analyst, I knew what a road looked like in a standard RGB image, I would still be

able to identify what a road was even if we took a similar image and mixed the representation of bands ($R \rightarrow G$, $G \rightarrow B$, $B \rightarrow R$ for example). I would not need pictures of labeled roads under this new scheme; I would know from features common between domains. This idea is the basis for this thesis. Can a CNN that learned what a road looked like in satellite imagery, be able to identify a road in aerial footage without labels?

CHAPTER 3: Methodology

3.1 Introduction

CNNs have proven themselves capable of rivaling human level performance on image classification tasks. Unfortunately, for many such classification tasks, large amounts of labeled data is required for this level of performance. This problem is precisely the issue faced while looking to train a CNN to identify roads from aerial imagery. Not only is aerial imagery not readily available, the process of labeling the data for training is manual and time consuming. This is even more true for aerial video feeds. While extensive, labeled training datasets for an aerial CNN are not available, we do have ready access to an abundant supply of current and archived satellite imagery and labeled road vector data. Using automated tools, we can quickly create a satellite imagery based CNN training dataset. This thesis evaluates the potential use of DANNs to leverage such labeled satellite data to train a CNN capable of identifying roads in aerial imagery without the expensive process of producing labels for that aerial imagery.

Previous chapters have explained the context and concepts required to understand this thesis. Later chapters will discuss the design, execution, results, and conclusions of this work. This chapter introduces the theoretical basis for everything that follows. We explain the overall theory of DA then discuss how to measure its limits. The limits of DA that we present are based on Domain Divergence. This is a measurable characteristic which we describe. Finally, we show how this measure of divergence, as it fits within setting the limits of DA performance, can be used as a basic structure for a new form of feedforward NN: DANNs. DANNs being the concept upon which this thesis is built around.

3.2 Domain Adaptation

Data is important to ML. It helps ensure model hypothesis more closely represent the true mapping between model input and output. While larger amounts of data can reduce this error, it does not generally produce a model capable of mapping beyond the specific domain

for which it was trained. More data from a domain can help a model generalize to a new domain, but not without error. This inability to generalize is what drives the creation of new models for each domain the model is to be used on, regardless of how much data from the original domain can be used. If enough labeled data for the new domain exists, then it is preferable to train a new model [2]. Unfortunately, the process of generating new, labeled data can be an expensive and time consuming process, preventing the creation of a new model [3]. Creating the dataset for this thesis alone, took well over 40 man-hours of dedicated time. Short-circuiting the cost of data-labeling is where DAs can provide an advantage.

3.2.1 Domain

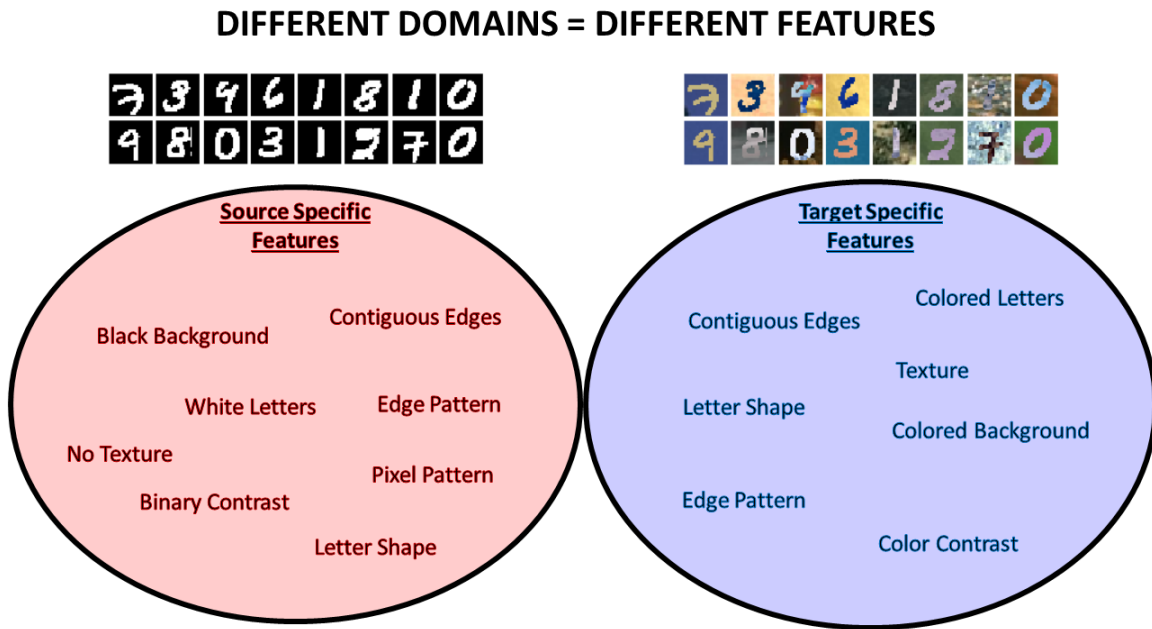
A domain, as defined by [2] for the purposes of discussing DA, is a labeling function $f : X \rightarrow [0, 1]$ paired with a matching distribution of data, $\mathcal{D}$, given as a set of inputs, X , to that function, $\langle \mathcal{D}, f \rangle$. See Figure 3.1 for an illustration. When we are referring to the source and target domains we will use the shorter notation $\mathcal{D}_S$ and $\mathcal{D}_T$, respectively. Some practical examples of domains include:

1. A specific email user and their determination that an email is spam or not spam.
2. A set of synthetic images and the function that created their synthetically created labels
3. A written review of a movie or book and the characteristics of that review that make it either positive or negative [2], [3].

3.2.2 Domain Adaptation

DA is a method for leveraging the labeled data of one or more domains (source domains) to train a discriminative classifier or predictor that will be evaluated against another domain (target domain, shifted from the original domain), that has little or no labeled data available [2], [3]. This process limits, or eliminates, the need to label enough training data to create a model for the target domain. Our only requirement for DA is that data used during training is from a similar distribution to the data our classifier will be evaluated on [3]. Applying DA to our examples from Section 3.2.1 we can see the benefit of not needing labeled data for an application similar to our original.

Figure 3.1. Notional source and target domains.



Notional source and target domain features and labels for MNIST and MNIST-M. Each domain is the distribution of data in the red or blue circle along with their labeling function. In this case, this labeling function says that if the data is within the red or blue circle, it is labeled source or target, respectively. In machine learning, we seek to find the boundary between the circles.

To train our spam filter we would train our model using all of the emails we previously marked as spam. We could then apply our spam filter, as is, to label spam for a new email user. Our new user would however experience a loss in performance relative to a spam filter trained solely on the type of spam/not-spam emails they received. This is because each user is likely to have different types of spam, or even have a different idea about what constituted spam [2]. In order to recoup that lost performance and train a user specific spam filter, we would have the user label a large number of emails as spam/not-spam before we had a dataset representative of his unique spam/not-spam distribution for training. Using a method of DA, however, we could leverage the labeled data available from old users to increase the performance of this new user's spam filter without requiring hundreds or thousands of manually labeled emails.

DA could, in theory, also be applied to the use of synthetically created imagery, fully

labeled, as a data source for a DA NN. Using DA and either no labeled, or some labeled target imagery, one could create an accurate CNN without having to label hours of images or video [3].

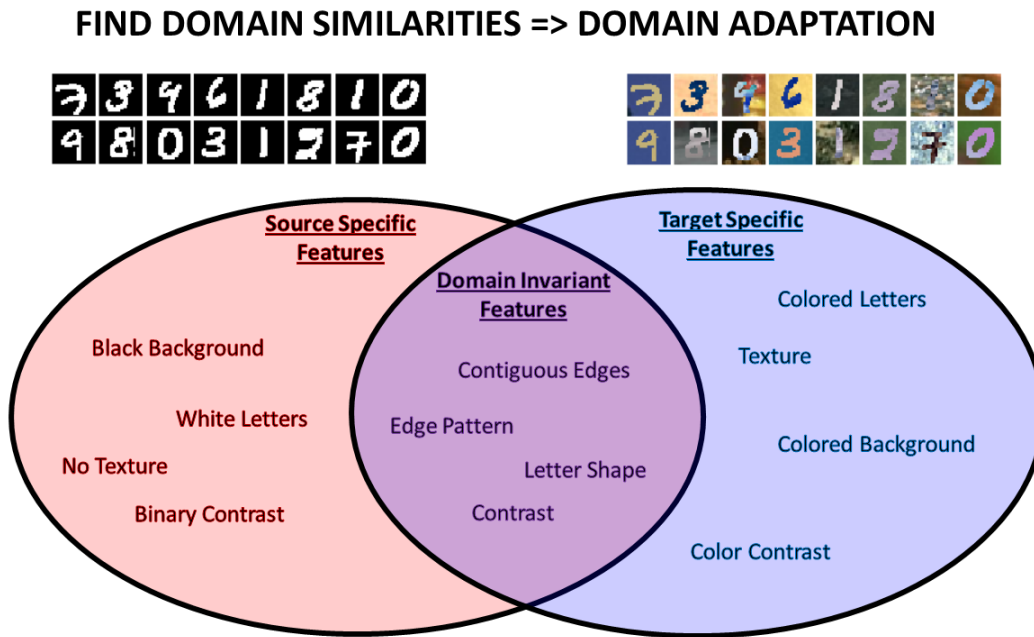
The performance limits of DA is bound by the performance of a source only, target only, or equally-weighted set of training data [2]. In other words, we cannot expect a hypothesis, created through DA to perform better than a source domain trained classifier classifying source domain data, or a target domain trained classifier classifying target domain data. This limit however is a worthy trade-off, as seen by state-of-the art performance benchmarks on sentiment analysis and image classification without labeled target data [3]. When data is being called “the new oil” by the former president of Google China [5], anything that can put data to use or increase its effectiveness is of value. Personally, I am all for a high performance spam filter that does not rely on hours of spam/not-spam labeling.

3.3 Domain Divergence

Understanding the distance, or divergence, between domains in DA is an important part of understanding the performance of DA hypotheses. Single domain hypotheses provide a base for performance comparisons. As a target domain begins to diverge further from the source domain, we can and do expect the performance of that source trained hypothesis to degrade when applied to that target domain [2]. This performance degradation is measurable, as shown by Ben-David et al. [2]. The key to an accurate measure of this becomes finding an accurate measure of divergence between source and target domains [3]. Such a measure is presented in [3] and is the theoretical underpinnings of DANNs: which this work is based on. See Figure 3.2 for a rough introductory illustration.

As an example of domain divergence, we can intuit that the distribution of spam/not-spam might be similar between the email accounts of Professor Alice and Professor Bob, who work for the same university department, attend the same conferences, with the same personal and professional interests. As we begin to look at the distribution of spam/not-spam received by professors in other departments or other universities, we can expect this distribution to diverge. At some point, the distributions of spam/not-spam will have diverged to the point that there is little to no similarity. The performance of a spam filter will steadily degrade as these distributions diverge. In other words, we can expect that a spam classifier, trained on

Figure 3.2. An illustration of domain adaptation.



There are several approaches to domain adaptation. Pictured is the general concept of DANNs applied to the MNIST/MNIST-M datasets. DANNs penalize features which can be used to distinguish domains (red and blue features), and gives more weight to features which are invariant between domains (purple features).

an NPS professor's distribution of spam/not-spam is not likely to perform well for a chicken farmer in Russia. This performance drop can be predicted with an appropriate measure of divergence in distributions and a baseline performance on the model trained and evaluated on the source spam/not-spam distribution.

3.3.1 $\mathcal{H}$ -Divergence and Theory

Statistics offers a number methods for determining the distance between data points and distributions. Each of these offers their own advantages/disadvantages for each potential application. Unfortunately, in this application, their disadvantages can include an overly restrictive measure of distance, as is the case with the L_1 distance [2]. In an effort to find an appropriate measure of divergence for classifiers, Ben-David et al., developed a new measure called $\mathcal{H}$ -Divergence [2].

Unlike other measures, $\mathcal{H}$ -Divergence is adapted specifically for a situation where an estimate of divergence must be measured from an unlabeled, finite set of samples from each domain [2]. In developing $\mathcal{H}$ -Divergence, Ben-David et al. provided proofs and experimental results to support this [2]. These results were based on training a linear model on a source domain and evaluating its performance on a target domain [2]. This showed that classifier-induced divergence can be estimated without labels.

$\mathcal{H}$ -Divergence is defined in [3]: “Given two domain distributions, $\mathcal{D}_S^X$ and $\mathcal{D}_T^X$ over X , and a hypothesis class $\mathcal{H}$, the $\mathcal{H}$ -Divergence between $\mathcal{D}_S^X$ and $\mathcal{D}_T^X$ ” is:

$$d_{\mathcal{H}}(\mathcal{D}_S^X, \mathcal{D}_T^X) = 2 \sup_{\eta \in \mathcal{H}} \left| \Pr_{X \sim \mathcal{D}_S^X} [\eta(x) = 1] - \Pr_{X \sim \mathcal{D}_T^X} [\eta(x) = 1] \right| \quad (3.1)$$

where $\mathcal{H}$ is a hypothesis class containing continuous or discrete binary classifiers ($\eta : x \rightarrow \{0, 1\}$) [3]. In other words, $\mathcal{H}$ -Divergence is the distance between source and target probability distributions of positively labeled inputs. The value of $\mathcal{H}$ -Divergence varies between 0, identical source/target domains, and 2, source and target domains that are completely divergent.

For illustration of $\mathcal{H}$ -Divergence, let's assume we are going to train Professor Bob's spam filter using Professor Alice's larger data set. Professor Alice has 1100 emails, three-fourths of which are spam — resort, medicine, and dating sites (a spammer got a hold of a conference registration list... supposedly). Professor Bob has 900 emails, but only a quarter of them are spam — resorts, medicine, and Nigerian Princes. Unfortunately, these 2000 emails are only a finite sample of the actual distribution of spam/not-spam for Alice and Bob and not representative of the true distribution, which cannot be known. For example, perhaps our sample was taken when our spammer is getting a unique kickback on clicks for resorts in Mexico, so our sample distribution does not show that both Alice and Bob are likely to also get spam for the Bahamas. We can work around this disparity between true and measured distribution though an empirical measure of $\mathcal{H}$ -Divergence.

The empirical $\mathcal{H}$ -Divergence is defined in [3] as:

$$\hat{d}_{\mathcal{H}}(S, T) = 2 \left(1 - \min_{\eta \in \mathcal{H}} \left[\frac{1}{n} \sum_{i=1}^n I[\eta(x_i) = 0] + \frac{1}{n'} \sum_{i=n+1}^N I[\eta(x_i) = 1] \right] \right) \quad (3.2)$$

where $I[a]$ is defined as an indicator function which produces a Boolean value: 1 if a is true and 0 if a is false, N is the combined total of source samples (n) and target (n') samples from each domain ($N = n + n'$), and S and T are defined as:

$$S = \{(x_i, y_i)\}_{i=1}^n \sim (\mathcal{D}_S)^n; \quad T = \{x_i\}_{i=n+1}^N \sim (\mathcal{D}_T^X)^{n'}. \quad (3.3)$$

Using our Alice (A) and Bob (B) spam example, $n = 1100$ and $n' = 900$. 75% of Alice's emails are spam while 25% of Bob's emails are spam. Let us assume, that we find a binary classifier, η , that correctly classifies all x_i in the source sum $\sum_{i=1}^n I[\eta(x_i) = 0]$ and all x_i in the target sum $\sum_{i=n+1}^N I[\eta(x_i) = 1]$. Our $\hat{d}_{\mathcal{H}}$ would be calculated as:

$$\hat{d}_{\mathcal{H}}(A, B) = 2 \left(1 - \left[\frac{1}{1100} 275 + \frac{1}{900} 225 \right] \right) = 1.0. \quad (3.4)$$

Now lets assume that our best η is unable to correctly classify any of Bob's Nigerian Prince spam, since our classifier was trained on Alice's resort, medicine, and dating site spam. If 40% of Bob's spam emails are Nigerian Prince emails, then 40% of his 225 actual spam emails will evaluate to 0 (135 evaluate to 1) in $\sum_{i=n+1}^N I[\eta(x_i) = 1]$. Our $\hat{d}_{\mathcal{H}}$ would now be calculated as:

$$\hat{d}_{\mathcal{H}}(A, B) = 2 \left(1 - \left[\frac{1}{1100} 275 + \frac{1}{900} 135 \right] \right) = 1.2. \quad (3.5)$$

With these two examples, we can see what can cause a change to $\mathcal{H}$ -Divergence. In the first example, the character of spam/not spam was the same and our classifier worked perfectly to distinguish between the two. Our two domains were still divergent, as the distribution of spam in each domain was different. We can also see $\mathcal{H}$ -Divergence increase in our second example due to a change in the data characteristics and resultant increase in classification error. This occurs because of the feature space induced internally by our classifier. This feature space has its own distribution [1].

Taking our spam example to each extreme, assume Professor Alice is actually Professor Bob on the weekends. Their separate email accounts both contain about 20% spam from

the same collection of government officials hoping to give you your lost money and home loan financiers. “Alice” has 600 emails and Bob has 1400 emails. η performs flawlessly.

$$\hat{d}_{\mathcal{H}}(A, B) = 2 \left(1 - \left[\frac{1}{600} 480 + \frac{1}{1400} 280 \right] \right) = 0. \quad (3.6)$$

The problem with calculating the empirical $\mathcal{H}$ -Divergence, as we just saw above, is that it requires labeled source data and *labeled* target data (to verify the indicator function $I[a]$). If we have enough labeled target data, we might as well train a new model on the target domain alone. The power that $\mathcal{H}$ -Divergence provides is not only in determining optimal mix of source and target domains, but also the performance of a source trained classifier classifying on an unlabeled target test domain (as in our spam case and research). For this, we extend our empirical estimate of $\mathcal{H}$ -Divergence further and use a measure call Proxy $\mathcal{A}$ -Distance.

3.3.2 Proxy $\mathcal{A}$ -Distance and Practice

The Empirical $\mathcal{H}$ -Divergence can be difficult to compute exactly, specifically due to the size of $\mathcal{H}$ (recall that this is defined as a hypothesis class containing continuous or discrete binary classifiers). The Empirical $\mathcal{H}$ -Divergence can be approximated instead through training a binary classifier which can tell the difference between source and target samples [2]. This approximated distance is called the Proxy $\mathcal{A}$ -Distance and its empirical value is defined by [3] as

$$\hat{d}_{\mathcal{A}} = 2(1 - 2\epsilon). \quad (3.7)$$

ϵ is the classification error of a model trained on a newly created, composite dataset. This new dataset U is composed of both source and target samples, labeled according to their domain, with target samples labeled 1 and source samples labeled 0:

$$U = \{(x_i, 0)\}_{i=1}^n \cup \{(x_i, 1)\}_{i=n+1}^N. \quad (3.8)$$

This new approximation, Proxy $\mathcal{A}$ -Distance, eliminates the need for labeled target data

to calculate the divergence between domains. More importantly, however, the Proxy $\mathcal{A}$ -Distance shows that we can determine domain divergence by identifying the characteristics (intermediate features in a learning algorithm) that separate the two domains. This distinction, training a model to classify based on all features vice just the source domain features, is vital to understanding the difference between $\mathcal{H}$ -Divergence and Proxy $\mathcal{A}$ -Distance.

We can see the similarity in Proxy $\mathcal{A}$ -Distance as the empirical estimate for $\mathcal{H}$ -Divergence $\mathcal{A}$ -Distance is defined as

$$d_{\mathcal{A}}(\mathcal{D}_S^X, \mathcal{D}_T^X) = 2 \sup_{A \in \mathcal{A}} \left| \Pr_{\mathcal{D}_S^X}(A) - \Pr_{\mathcal{D}_T^X}(A) \right|. \quad (3.9)$$

In this form, $\mathcal{A}$ -Distance and $\mathcal{H}$ -Divergence are identical if $\mathcal{A} = \{A_\eta | \eta \in \mathcal{H}\}$ and A_η is the range of features produced by η .

This concept of using a domain classifier to determine $\mathcal{H}$ -Divergence through Proxy $\mathcal{A}$ -Distance is best illustrated again by our spam example. Lets first consider our example from equation 3.4 where our spam/not-spam examples were the same for Alice and Bob, but their distribution of spam/not-spam alone diverged. We can expect the optimal domain classifier to learn to label all spam emails as belonging to Alice and all not-spam emails as belonging to Bob as this configuration does the best to reduce our error during training (Remember that Alice had 75% spam and Bob had 75% not-spam). Using this classifier, we can expect that 25% of Alice's not-spam emails will be miss-classified as Bob's emails, and 25% of Bob's spam emails will be classified as Alice's emails. Our total error is 25% and our Proxy $\mathcal{A}$ -Distance is 1:

$$\hat{d}_{\mathcal{A}} = 2(1 - 2(0.25)) = 1. \quad (3.10)$$

This matches our $\hat{d}_{\mathcal{H}}(A, B)$ from equation 3.4.

Our Nigerian Prince spam emails help illustrate how the Proxy $\mathcal{A}$ -Distance allows calculation of $\mathcal{H}$ -Divergence without labels. For our Nigerian Prince $\mathcal{H}$ -Divergence example, our classifier was trained on the source samples only and required labels to verify the indicator

function, $I[a]$. For our Nigerian Prince Proxy $\mathcal{A}$ -Distance calculations, we train a classifier on *ALL* features available in the source and target domains. Our domain classifier, with the example distribution, will learn to classify all normal spam emails as belonging to the source domain, and all Nigerian Prince and not-spam emails as belonging to the target domain. This arrangement minimizes the classifier error. Evaluating U , our combined dataset, our classifier will miss-classify 25% of the source domain and 15% of our target domain. Our total error is 20% and our Proxy $\mathcal{A}$ -Distance is 1.2:

$$\hat{d}_{\mathcal{A}} = 2(1 - 2(0.2)) = 1.2. \quad (3.11)$$

This matches our $\hat{d}_{\mathcal{H}}(A, B)$ from equation 3.5. From this correlation between domain classification and $\mathcal{H}$ -Divergence we are able to identify the pivotal component of DA presented by [1]–[3] on which DANNs are built.

3.3.3 Target Error, Source Error, and Domain Divergence

Ben-David et al. prove, and demonstrate experimentally, an upper limit to the error to be expected by a source-domain trained classifier evaluated on a target-domain [2]. This limit to target error, $\epsilon_T(h)$, is dependant on, as shown in equation 3.12:

1. The empirical source error, $\hat{\epsilon}_S(h)$.
2. A complexity error term for the empirical measure of source error, $\frac{4}{m} \sqrt{\left(d \log \frac{2em}{d} + \log \frac{4}{\delta}\right)}$.
3. The empirical $\mathcal{H}$ -Divergence, $d_{\mathcal{H}}(\tilde{\mathcal{U}}_S, \tilde{\mathcal{U}}_T)$.
4. A complexity error term for the empirical $\mathcal{H}$ -Divergence, $4\sqrt{\frac{d \log(2m') + \log(\frac{4}{\delta})}{m'}}$.
5. The error from the ideal joint hypothesis on source and target domains, λ .

Together, this upper bound is defined as:

$$\epsilon_T(h) \leq \hat{\epsilon}_S(h) + \frac{4}{m} \sqrt{\left(d \log \frac{2em}{d} + \log \frac{4}{\delta}\right)} + d_{\mathcal{H}}(\tilde{\mathcal{U}}_S, \tilde{\mathcal{U}}_T) + 4\sqrt{\frac{d \log(2m') + \log(\frac{4}{\delta})}{m'}} + \lambda. \quad (3.12)$$

The complexity terms are dependent on the Vapnik-Chervonenkis (VC) Distance, d , which is defined roughly as the capacity of a learning algorithm to represent complexity [22]. For $\mathcal{H}$ of fixed complexity, these (ii and iv) remain constant. λ also remains constant (defined as β in [3]). Both concepts are referenced or explained in [1]–[3]. $\tilde{\mathcal{U}}_S$ and $\tilde{\mathcal{U}}_T$ are samples of the source and domain features-space induced by the domain classifier. Equation 3.12 varies in presentation between [1]–[3]; this version emphasizes the important underlying focus for DANNs.

In bounding the target error, we see that we are able to effect the performance of our source trained target classifier on the target domain by finding a hypothesis that minimizes $f(X) - h(X)$ (i.e., $Y \neq h(X)$) and by minimizing the $\mathcal{H}$ -Divergence between source and target domain features [3].

3.4 DANNs

A DANN, as presented by [3], is a modification to existing deep learning frameworks that implements a trade-off between error on the source domain and a generalization of features used by that framework. By using this modification, the required deep feature mapping, label prediction, and domain “convergence” can be completed using existing methods for deep learning in one training process. Under the hood, DANNs perform label prediction just like any other feed forward NN. Specifically, raw inputs are fed into a network of perceptrons that generate, through back-propagation, the deep features that optimize the performance of the final label predictor. This optimizes the performance of the label predictor as the first half of the aforementioned trade-off.

The unique addition to the DANN architecture is a domain classifier which operates in parallel to the standard label prediction process. As deep features are defined and fed into a label predictor, these deep features are also fed into this domain classifier. Ganin et al., as shown in Section 3.3.3, establish that reducing $\mathcal{H}$ -Divergence reduces target error. This domain classifier essentially determines for the DANN how much $\mathcal{H}$ -Divergence is present across the deep features induced while training the label predictor. Through back-propagation and a Gradient Reversal Layer (GRL), this feedback allows the DANN to “forget”, or un-weight, the domain specific deep features which cause the source and target domains to diverge.

It is important to note how DANNs, while implementing the work of Ben-David et al., creates a trade-off between source domain accuracy and a reduction in induced domain divergence [2]. The performance of any NN is dependent on its ability to identify features, within its intended domain, that allow its predictor to separate the examples provided to it. DANNs improve their performance on their target domain, by forcing the model to “unlearn” the deep features used to classify the source domain only. Since the DANN label predictor has fewer deep features to rely on for labeling the source domain, we can expect a reduction in performance on the source domain.

3.4.1 Domain Classification and the GRL

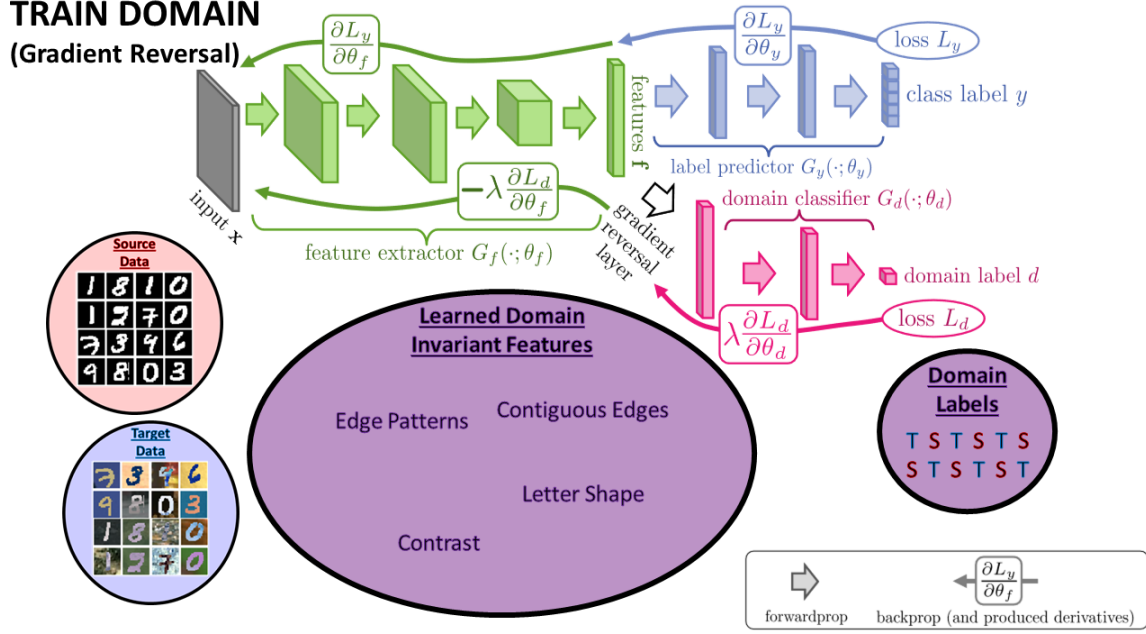
While seemingly complex, DANN implementation can be boiled down to a domain classifier and a “minus one” (internal to the GRL). The GRL is placed between the domain classifier and the deep features common to it and the label predictor [3]. During the forward pass of data, features remain unchanged heading into the domain classifier, as the loss from this forward pass is back-propagated, the gradient is reversed by a negative hyperparameter set prior to training. This GRL is the DANN component that directly computes the $\mathcal{H}$ -Divergence and implements the reduction in $\mathcal{H}$ -Divergence as proposed by [1]–[3]. It reduces the weights of features that allow for better domain classification.

In other words, this implementation is essentially a regularizer that penalizes divergence of deep features from each domain. The loss resulting from each forward pass into deep features on through the domain classifier can be input directly into our equation for Proxy $\mathcal{A}$ -Distance as $2(1 - R(W, b))$ [3]. This regularizer is defined as:

$$R(\theta_f, \theta_d) = -\max_{u,z} \left[\frac{1}{n} \sum_{i=1}^n \mathcal{L}_d^i(\theta_f, \theta_d) + \frac{1}{n'} \sum_{i=n+1}^N \mathcal{L}_d^i(\theta_f, \theta_d) \right] \quad (3.13)$$

where θ_f is the set of deep features common to the gradient regularizer and the label predictor, and θ_d are the weights of the gradient regularizer itself (See Figure 3.3).

Figure 3.3. Domain classifier training of a DANN. Source: [3]



Illustrated is the MNIST/MNIST-M DANN during its domain classifier training. The important part of this illustration is that the feature layer is learning domain invariant features due to the domain classifier and gradient reversal layers.

3.4.2 DANN Optimization Functions

DANNs are optimized by adjusting loss to support the objectives outlined in the theories of Ben-David et al. [2]. As seen in the equations below, a DANN's overall loss is the combined loss from label prediction and the negative weighted loss from the gradient classifier [3]:

$$E(\theta_f, \theta_y, \theta_d) = \frac{1}{n} \sum_{i=1}^n \mathcal{L}_y^i(\theta_f, \theta_y) - \lambda \left(\frac{1}{n} \sum_{i=1}^n \mathcal{L}_d^i(\theta_f, \theta_d) + \frac{1}{n'} \sum_{i=n+1}^N \mathcal{L}_d^i(\theta_f, \theta_d) \right) \quad (3.14)$$

where θ_y is the set of weights for the label predictor. Note that λ is used differently between [2] and [3]. Gradient descent can progress normally with the following updates where the learning rate is denoted as μ and the GRL regularization weight is denoted as λ [3]:

$$\theta_f \leftarrow \theta_f - \mu \frac{\partial \mathcal{L}_y^i}{\partial \theta_f^i} + \mu \lambda \frac{\partial \mathcal{L}_d^i}{\partial \theta_f^i} \quad (3.15)$$

$$\theta_y \leftarrow \theta_y - \mu \frac{\partial \mathcal{L}_y^i}{\partial \theta_y^i} \quad (3.16)$$

$$\theta_d \leftarrow \theta_d - \mu \lambda \frac{\partial \mathcal{L}_d^i}{\partial \theta_d^i} \quad (3.17)$$

3.4.3 Nigerian Princes

Our spam example continues to work well for illustrating how DANNs work. Lets first consider the case where we use an unmodified deep NN trained on Alice’s emails, then evaluate it against Bob’s emails (which include emails from Nigerian Princes). This unmodified NN would learn deep features unique to Alice’s email distribution that allow it to label an email as spam or not spam. Remembering that Alice receives resort, medicine, and dating sites spam we might expect deep features to activate on words or phrases like: savings, money, doctor, pills, magic, secret, info, exclusive discount, limited supply, act fast, do not wait, members only, personal match, offers, holiday, dream vacation, etc. Since Bob receives resort, medicine, and Nigerian Prince spam, we can expect deep features like: lost money, kindly, wire transfer, etc. to not activate since they were never learned.

As we saw when calculating our Proxy $\mathcal{A}$ -Distance for Alice and Bob, the addition of Nigerian Prince spam increased our divergence between domains. We need a way to reduce divergence. Modifying Alice’s email filter into a DANN, we add a gradient classifier that learns which features distinguish the domains and penalize them during training. During this process, we can expect deep features associated with dating sites to lose weight and more generic spam features to grow in influence. However, we cannot expect the unique features of Nigerian Prince spam to be learned as features of spam, since we do not provide Nigerian Prince labels to the DANN label predictor. If we wanted to promote activation on these deep features, we would need to provide labeled examples of Nigerian Prince spam.

3.4.4 MNIST DANN Example

In [3], DANNs are evaluated against a number (pun intended) of data sets. This includes the well known MNIST data set from Yann LeCunn et al.’s 1998 paper, “Gradient-based learning applied to document recognition” [23] which includes thousands of labeled black and white images containing the digits zero through nine. For this data set, and others, a lower and upper bound to DANN performance was established. This was done through training two models that were evaluated on target data: the first model was a source-only trained CNN (lower bound), the second was a target-only trained CNN (upper bound). These bounds represent performance when no DA is attempted and highlight the potential value of DA.

The target domain, labeled MNIST-M, was created by combining MNIST with another data set called BSDS500 from Arbelaez et al., 2011 [24]. BSDS500 is a collection of color photos. These provided image clips that were inverted, pixel-by-pixel, with MNIST essentially acting as a binary mask. The operation of this transformation is described in [3] for these two data sets as:

$$I_{ijk}^{out} = |I_{ijk}^1 - I_{ijk}^2|. \quad (3.18)$$

I^1 and I^2 are images from each data set and i, j, k are the x, y coordinates (i, j) for each pixel and its corresponding channel (k). The newly created data set, MNIST-M remains labeled through the individual MNIST images that created the composite one. MNIST-M presents a challenge for standard CNNs, but is not a challenge for human interpretation. A sampling of both the MNIST and MNIST-M dataset is displayed in Figure 3.4.

The DANN created for the MNIST to MNIST-M was kept simple for computation time. Its feature extraction component consisted of two convolution/pooling layers. The label classifier contained two fully-connected/ReLU layers followed by a fully-connected layer with a softmax classifier. The domain classifier was even simpler, containing only one fully-connected/ReLU layer and a fully-connected layer that fed into its final logistic classifier. [3] noted that the structure of the DANN was kept simple intentionally and that the author expected better performance with more architectural tuning.

λ , the domain adaptation parameter for the feature extractor was set to increase from 0 to 1

Figure 3.4. Sample MNIST/MNIST-M clips. Source: [3], [16].



The top row of images are from the original MNIST dataset. The bottom row is the MNIST-M dataset following the transformation effected on the MNIST dataset using equation 3.18.

under the schedule:

$$\lambda_p = \frac{2}{1 + \exp(-\gamma p)} - 1. \quad (3.19)$$

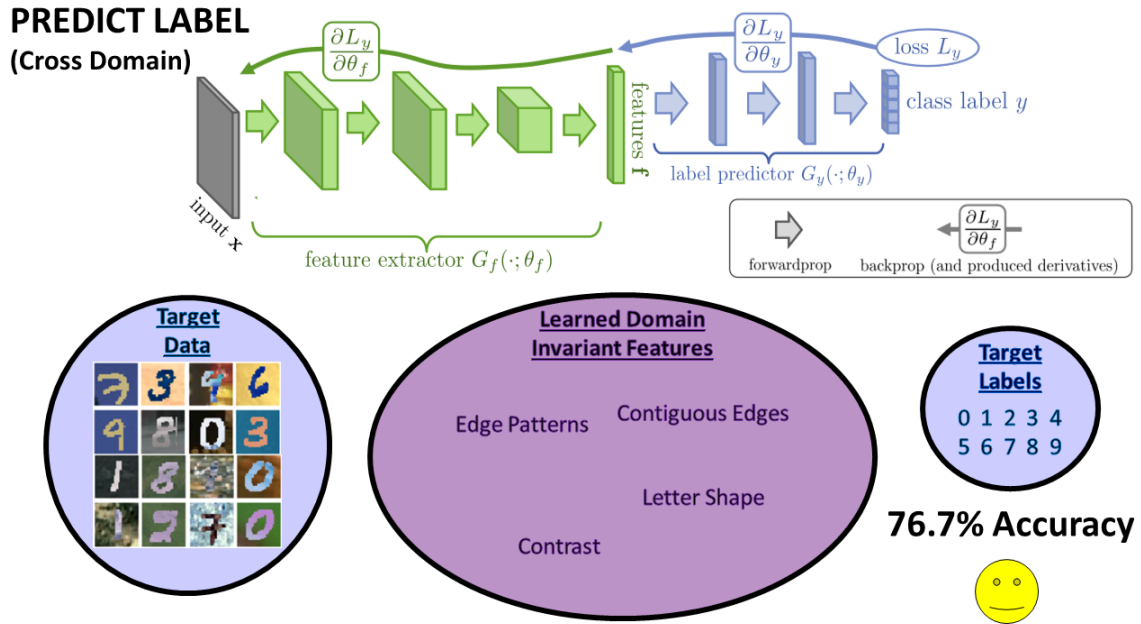
λ was set to 1 for the domain classification portion of the DANN. Each batch used for SGD is 128 samples. The first half of the batches contained source domain data with labels. The second half contained target only samples without (revealed) labels [3].

The results of this DANN show that they have quite a bit of utility, especially considering that the model architecture used was not optimized to the task. The classification accuracy for target domain data on the source-only trained model was a mere 52.25% (lower bound). The target trained model had a classification accuracy of 95.96% on target data (upper bound). Using a DANN, classification accuracy increased from the lower bound to 76.66%. This simple implementation closes the gap between lower and upper bounds by 52.9%. See Figures 2.1, 2.2, 3.3, and 3.5. The accuracy in these figures are those mentioned above.

3.5 Summary

The labeling of data, in any amount or type, can be a time consuming process and is prone to error. This is no different for our target domain: aerial video. This is particularly true for the author of this thesis, having had to manually label enough target domain data to validate the results of this thesis. Without using such labels, DANNs, in their originating research, demonstrated state of the art performance in DA against marginalized Stacked

Figure 3.5. MNIST/MNIST-M DANN. Source: [3].



Illustrated is the MNIST/MNIST-M DANN, post training, labeling data. Model performance increases after including the domain classifier in the training process as shown in 3.3.

Autoencoders (which attempt DA from a different approach) [3].

The DANN approach to DA without labels is accomplished through “promote[ing] the emergence of feature that are (i) discriminate for the main learning task on the source domain and (ii) indiscriminate with respect to the shift between the domains” [3]. Unlike other methods, DANNs have been designed to accomplish these separate processes of DA (ii) and feature learning (i) at the same time [3]. Comparably, DANNs are desirable for their simplicity and performance both for this application and in environments where resource availability for labeling target domain samples might not be feasible.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 4:

Design

4.1 Introduction

DANNs are designed to reduce the requirement for labeled data needed to train a classifier on a domain different than the one it is to be employed on. This creates a tremendous advantage when applied to domains where labeled examples are not readily available. This research explores the applicability of this advantage to a remote sensing setting, where the source domain is labeled satellite imagery and the target domain is unlabeled UAV imagery.

Previous chapters have reviewed the theoretical underpinnings of DANNs. This chapter outlines our experimental design. We review the setting for our research, experimentation, tools, data production and sampling, measurements, procedures, and methods for analysing our results. Subsequent chapters will discuss experimental results, then later an analysis of those results, limitations of our work, and areas for further exploration.

4.2 Setting

Data collection for this study centers on Camp Roberts, California; the primary research, test, and evaluation site for NPS's Center for Autonomous Vehicle Research (CAVR). Camp Roberts sits halfway between San Francisco and Los Angeles, straddling the border between Monterey and San Luis Obispo Counties. Nearby towns are small, with rural homes greatly dispersed. Most of the region is used for ranching and farming with most of the roads throughout the area unpaved.

Camp Roberts itself had been used for ranching and growing of grain from the founding of Mission San Miguel de Archangel 1797 until its acquisition by the U.S. Army in 1943. During its time as an active Army base, Camp Roberts was used for everything from basic training of infantry to tanks and artillery and supported as many as 45,000 troops at its peak. At one point it was home to the U.S. Army 7th Armored Division and was (and is) used for vehicle and weapons testing. The site was closed as a U.S. Army base in 1970 and turned over to the California Army National Guard [25].

Covering an area of nearly 43,000 acres in the Coastal Range of Central California, it is one of the largest Army training installation in the U.S.. It also covers a large variety of ecological zones and vegetation as seen in satellite imagery, in person surveys, and Berkeley’s Wieslander Vegetation Type Mapping database [26]–[28]. Specifically, this area is a mix of grass rangeland, scrub, and trees. In addition to normal vegetation, large tracts of the Camp Roberts training area appears burnt, changing the appearance of normally golden brown grassland to a mix black, grey, and white ash (likely due to controlled burns as part of a responsible range management program). See a satellite overview of the region in Figure 4.1.

A number of road types are visible in the satellite and UAV video to be used. Paved roads measure approximately 3-12 meters wide with a light to dark grey appearance. They typically have clear boundaries with the road shoulder and contrast well with the surrounding terrain. Dirt and gravel roads are common throughout both datasets. Dirt roads are generally 2-3 meters wide and light brown to bright grey in appearance. Their shoulders are less well defined, but still contrast well with their surroundings. "Jeep trails" are present as well, appearing as two discernible, parallel tracks through open areas. These trails are generally 2-3 meters wide are generally lighter than the surrounding terrain, and often meander or loop back on themselves. "Game trails" are also visible. These are narrow single track paths through open/grassy areas. Also see 4.2.

While paved and unpaved roads are easily discernible, even through more dense forested/treed areas, Jeep and game trails can require a substantial amount of analyst interpretation to identify and track. This ambiguity in track and class can lead to error and realistically should exist as their own class in the generated datasets (i.e., Road/Trail/NotRoad vs Road/NotRoad).

4.3 Experimentation

This research seeks to quantify the benefits of DANNs, relative to NNs that are not domain adapted, on the specific task of predicting the class of items in a target dataset when no labeled target data is available for training. In the work of [2], we have the theoretical bounds of performance for such a classifier. At its lowest, our classifier will do no worse than a NN trained on satellite data, classifying ScanEagle data (no domain adaptation).

At its best, our classification task will perform no better than a NN trained on ScanEagle data, classifying ScanEagle data (perfect domain adaptation, since it is adapted to itself). These bounds provide the fixed performance range which we are to identify the optimal performance of our DANN.

This research will identify upper and lower bounds using non-domain adapted NN for satellite and UAV video. DANN classification accuracy will be evaluated against these performance bounds. We expect results to vary with changes in DANN architecture, specificity of domain definitions, and methods for data pre-processing (input image size, etc). We will only be evaluating changes in performance due to architecture.

4.3.1 Model Architecture Building Blocks

Stanford’s CS231n compares building NN to building Legos [7]. Each block has a specific “function” and fits with other blocks in a very specific way. Individual blocks can be built into bigger “blocks” with their own “function.” Throwing blocks together randomly might make for a pretty ugly rocket, but bringing together the right pieces in the right order can make for something pretty spectacular. The blocks used for our three base model architectures are illustrated in Figure 4.2.

4.4 Materials

NN are data hungry and computationally dense. They typically perform only as well as the quality of inputs and tools available. For this research, models are to be fed with data from NPS ScanEagle video and satellite imagery obtained from DigitalGlobe. All tools, excepting ArcGIS Pro, are readily available or free for use. The licence to use ArcGIS Pro for this research was acquired through NPS, though it is also available at a reduced rate for students through Esri.

4.4.1 Data Sources

NNs generalize well when trained with quality inputs that are representative of the data they are meant to compute an output for. Data for our specific use case does not meet this criteria due to an asymmetry in data availability: the reason for our evaluation of DANNs. Large repositories of quality satellite data are available through companies such as DigitalGlobe

or organizations such as the National Geospatial-Intelligence Agency (NGA). This is not the case for UAV imagery.

The characteristics of the data itself will be described more thoroughly in Section 4.5.

Satellite

All satellite data was acquired through a company called DigitalGlobe. DigitalGlobe is a company that owns and operates a constellation of five high-resolution, panchromatic to super-spectral satellites. They provide satellite imagery to a variety of customers including the United States Government (USG), telecommunications, automotive, and members of the global defense and intelligence community [29], [30].

Satellite data for this research was downloaded from DigitalGlobe's EnhancedView Web Hosting Service (EV-WHS) through a DoD subscription. EV-WHS is a web-based application that provides the ability to perform basic data analysis, viewing, query, and construction of satellite datasets [31].

Road Labels

Initially, road vector data was downloaded for use from NGA. NGA is a government agency that provides geospatial intelligence to the U.S. intelligence community and the DoD. NGA produces and distributes a wide array of data to include vector and raster maps, elevation data, satellite imagery, bathymetric data, aviation maps with in-flight hazards, etc.

Road vector data from NGA was intended as a source of labels for the satellite imagery downloaded from EV-WHS. This vector data from NGA was quickly discarded due to issues with the data being out of date and trouble matching map projections between the road vector data and the satellite data. Road vectors from this source, as loaded, were often up to 70 meters off of satellite road center line. They also did not cover areas of recent road development. Satellite imagery acquired from DigitalGlobe was recent within the last few years, where-as data from NGA was from a different decade entirely.

To eliminate the extra source of variability and error, road vector data for Camp Roberts was created manually using ArcGIS Pro.

Aerial

UAS video footage was acquired from a ScanEagle flown by NPS CAVR over Camp Roberts. The ScanEagle was being controlled manually by a drone operator from the UAS ground control station (GCS). The ScanEagle did have a secondary controller on-board that ran the open-source Robot Operating System (ROS). ROS managed the collection and transmission of data. Data from the collection flight was stored in a ROS bag file and video was stored in a packet capture (PCAP) file prior to being extracted for use.

4.4.2 Hardware

Data and ML model characteristics are very much determined by the capabilities and limitations of the collection, transmission, and pre- and post-processing platforms used.

Satellite

Imagery for this research was acquired by DigitalGlobe's WorldView-2 (WV02) Satellite. WV02 has been in orbit since October 2009 and is currently one of five satellites used by DigitalGlobe. It is able to collect multispectral imagery across 8 spectral bands (400-450 nm, 450-510 nm, 510-580 nm, 585-625 nm, 630-690nm, 705-745 nm, 770-895 nm, and 860-1040 nm) as well as panchromatic imagery (450-800 nm). Resolution for multispectral imagery from WV02 is as good as 1.85 meter GSD. Panchromatic imagery from WV02 is available at a much higher resolution: 0.46 meter GSD at nadir. Resolutions are slightly lower off-nadir [32].

Imagery is transmitted through an 800 Mbs X-band data link from WV02's sun synchronous orbit at 770 kilometers above sea level. At this altitude, WV02 is capable of collecting 1 million square kilometers of imagery per day. It is also able to revisit an area every 1.1 days [32].

ScanEagle

The ScanEagle UAS is a product of Insitu. The aircraft is a small, modular UAV with and a rear-mounted pusher propeller. Depending on its specific configuration, the ScanEagle is about 1.5 meters long with a 3 meter wing span. It can carry a variety of payloads with a max takeoff weight of 18 kilograms (12 kilogram empty weight). The ScanEagle system

includes a GCS, launch system, and retrieval system. It can be configured for both sea or land operations [33].

The ScanEagle standard camera payload (Sony EX780) provides color video with a 640x480 resolution at 30 frames per second and a 1.8° to 45° FOV. This camera is mounted to an inertially stabilized turret in the aircraft nose that can point to the sides, down, front, and back. Video is transmitted to the ground receiver through an analog L-band link [33].

Computer

The computer to be used for the majority of data processing and model training and test for this research is a custom built desktop. Intended for ML, the system is built around an NVIDIA GeForce RTX 2080 Ti GPU. Supporting the GPU is an AMD Ryzen 7 3700X 8-core processor, and 16GB of DDR4-3600MHz memory. Data is stored on the system in a 1TB PCIe NVMe M.2 solid state drive. A second GPU (NVIDIA GeForce 1660) is used for video display, and other applications, to free up the RTX 2080 Ti entirely for ML applications.

4.4.3 Software

The computer system used for this research ran Microsoft Windows 10 Pro. While interaction with ROS and TensorFlow 2.0 would have been easier (and faster as tested through informal benchmarks) with Ubuntu Linux, ArcGIS Pro 2.2 is available for Windows only. For ease of access to the data and visualization, Linux was not used. All code was generated and tested with the Anaconda distribution of Python 3.6 and 3.7, is stored on the NPS GitLab server, and is available for review upon request.

Data Preprocessing

ArcGIS Pro, by Esri, is a Geographic Information System (GIS) application. GIS is essentially a framework that links data to a geographic location. This allows an analyst to visualize layers of data according to how it relates spatially. In simpler terms, GIS allows an analyst to create maps of data to identify spatial trends and scenes [34], [35]. ArcGIS will be used to analyse satellite data for Camp Roberts, to create an accurate road dataset, and to extract satellite clips for training using ArcGIS Pro's python scripting API.

Video data was acquired and stored embedded in a PCAP file. Wireshark is used to identify, extract, and reassemble MPEG packets into a video file for subsequent processing. These video files were then processed with Python scripts using OpenCV, ImageIO, and Numpy to manually select a variety of nadir to low-oblique video frames absent severe transmission artifacts. These frames were then processed into clips, manually labeled (for DANN performance validation), and sized roughly to match the scale of data found in the satellite data.

Once raw, labeled clips were generated, scripts sorted them further into a file structure friendly to TensorFlow's ImageDataGenerator class `flow_from_directory()` method.

Model Building, Training, and Evaluation

TensorFlow is a software platform developed by Google to facilitate machine learning models. Keras is the high level API adopted by TensorFlow to increase usability and speed model development [36]. The newer TensorFlow 2.0 for GPU is used for all deep learning for this research. Custom layers (GRL, etc) and models were created using the Keras Model and Layer Subclassing API. Frozen features for transfer learning were downloaded for Inception/Resnet V2 using Keras as well. Model results are processed and visualized using the Python Matplotlib and Numpy packages.

4.5 Data

There are three primary sources of data for this research: satellite imagery, road vectors, and UAS video. Raw satellite imagery and road vectors will be used to generate the labeled source domain dataset for identifying the lower bounds of DANN performance and later training the DANN. UAS video will be used to generate the unlabeled target domain dataset (labeled for validating performance). This dataset will be used to identify the upper bounds of DANN performance and later for training the DANN. Details of dataset creation will be covered in Section 4.6.

Table 4.1. Satellite imagery metadata.

Date and Time (UTC)	Cloud Cover	Off Nadir
2019-07-24 19:09	0%	21°
2018-09-20 19:03	0%	7°
2017-08-16 19:11	0%	26°
2017-07-12 18:59	4%	9°
2019-04-02 19:11	0%	17°

This table displays the date/time, cloud cover, and degrees off nadir of images used for the satellite mosaic imagery.

4.5.1 Description

Satellite

The color satellite imagery for Camp Roberts was constructed as an imagery mosaic of data from five different WV02 collection passes. Cloud cover on each date of collection was 4% or less and an average of 16° off nadir (see Table 4.1). All imagery has a GSD of 0.5 meters or less. Imagery used only contains three channels: red, green, and blue. Road data was manually generated using ArcGIS Pro and stored as linear vectors in an ArcGIS shapefile.

Aerial

UAS video footage was collected from an NPS CAVR ScanEagle on 14 September 2018. This collection flight generated just short of an hours worth of 640x480 pixel footage at 30 frames per second. During video collection, the ScanEagle flew in rough, parallel tracks across Camp Roberts at 3,500 and 5,000 feet MSL (approximately 2,500 and 4,000 feet AGL). By comparing road width in pixels between satellite imagery and ScanEagle video, we can estimate the ScanEagle video to be at approximately 2 meter GSD as compared to the satellite imagery's 0.5 meter GSD (see equation 4.1 and Figure 2.7).

$$\begin{aligned}
12pixels \left(\frac{0.5m}{pixel} \right) &= 6m \\
\Rightarrow \frac{6m}{3pixels} &= \frac{2m}{pixel}
\end{aligned} \tag{4.1}$$

Since video from the ScanEagle is transmitted in analog, quality varies throughout the flight depending on distance from the GCS, angle of bank, and signal interference. During turns, features often blur, but regain sharpness once the ScanEagle stabilizes. Throughout most of the video, the ScanEagle Video Sensor is pointed nadir to low-oblique with occasional transits to high-oblique during turns.

4.5.2 Sampling Method

For training the CNNs and DANNs, the larger satellite mosaic and UAS video feed was broken into labeled clips with dimensions that were factors of the ScanEagle 640x480 video dimensions (to make later math easier). Two satellite clip sets were made with clip size 120x120 and 160x160. One UAS video clip set was manually created with clip size 40x40. This process is described more thoroughly in Section 4.6.

All four datasets were broken into training, validation, and test sets using a 50/25/25 percentage split. Clips were randomly selected, without replacement, from the original pool of clips using Python’s `random.sample()`.

4.6 Procedures

Without access to labeled UAS video clips, labeled satellite clips, and having a desire to remove variability introduced from existing road vector data, a substantial portion of this research effort was spent on the creation of these datasets. Following creation of these datasets, various model architectures were trained and evaluated against pure satellite or ScanEagle datasets to generate the upper and lower theoretical bounds for DANN performance. Once these bounds were determined and the most performant architecture was identified, we proceeded with training and tuning the DANN. Throughout this process, the MNIST/MNISTM datasets from [3] was used to validate that our model architectures and

code are working as intended.

4.6.1 Phase I – Define Domains and Build Datasets

If we are to expect our NNs to learn to classify roads and trails within our domain, then we need clear definitions of what constitutes a road or trail within our domain. To simplify the labeling process, all roads and trails will be classified as “roads” and anything else classified as “not-road.”

The larger road domain includes a number of other characteristics common across all “sub-domains” (paved, unpaved, etc) that help with identification. Roads or trails generally have at least one end attached to either another road, trail, or other man-made or natural point of interest, i.e., they serve a purpose. Conversely, if a man-made structure is present in a scene, then a road or trail is also likely nearby.

Imagery was acquired from DigitalGlobe and a ScanEagle flown over Camp Roberts and/or the surrounding areas. To remove variability, imagery from both sources will be collected between nadir to low-oblique. Low cloud cover is desirable as are acquisition months similar to one another (where available) to eliminate potential changes in road appearance with annual variance in vegetation.

To remove variability, imagery was selected from both the DigitalGlobal satellite and ScanEagle that was between nadir to low-oblique. Additionally, data was collected from both platforms during similar times of year in order to eliminate potential changes in road appearance with annual variance in vegetation.

ArcGIS Pro was used to evaluate road vector data from NGA or to manually create road vector data according to the descriptions presented in 4.2. Subsequent processing used the ArcGIS Pro Python scripting API. Specifically, these road vectors were converted to polygons centered on a small buffer around each vector, sized to match average road/trail width. These road polygons were then input into the API’s `ExportTrainingDataForDeepLearning()` tool with the target imagery. The output of this tool is a set of clips with matching text files containing KITTI labels (format described at [37]).

Using the KITTI labels and clip pixel data, clips are filtered. Clips are discarded if the clip KITTI label indicates only a marginal portion of the clip contains a road and if black pixels

Table 4.2. Domain class definitions.

Label	Description	Width (m)
Paved Road	Light to dark grey appearance. Often has visible, painted center-lines and shoulders, clear boundaries at the road shoulder, and good contrast with surrounding terrain.	3-12
Unpaved Road	Light brown to bright grey in appearance. Shoulders are less well defined, but still contrast with their surroundings.	2-3
Jeep Trail	Appear as two discernible, parallel tracks. These trails are generally lighter than the surrounding terrain, but in certain areas blend so well that they are identifiable by texture differences only. They often meander or loop back on themselves.	2-3
Game/Foot Trail	These are narrow single track paths that are only identifiable where they contrast with the terrain they pass through.	.5-1

Definitions of road and trail classes within the source/target satellite/UAS imagery domains. All of these classes are joined into one “road” class for training. Everything else is classified as “not road.” See Figure 5.1 for samples from the spectrum of “roads” within our 160x160 satellite imagery domain.

are present (an artifact of ArcGIS clipping at the borders of imagery).

ScanEagle video is extracted from saved PCAP files using Wireshark. The Python ImageIO package and OpenCV is used to review and save individual frames from this video. Frames are saved only if they display a unique scene, are nadir to low-oblique, are free of major video transmission errors, and relatively blur free. These frames are then broken into overlapping clips and labeled clip-by-clip as “road” or “not road.”

Both datasets are then sampled randomly, without replacement, to be included in a training, validation, or test set in a 50/25/25 split. Clips are then saved in a file structure required for TensorFlow’s ImageDataGenerator class flow_from_directory method. Test sets will

Table 4.3. Non-DANN architectures

Model	Description
Simple 6-layer CNN	Three convolutional blocks followed by two dense layer blocks (our standard label classifier). See Figure 4.3.
Custom, abbreviated Inception/ResNetV2-ish CNN	Highly modified model based on Inception/ResNetV2. Composed of an initial stem designed for capturing base features with complexity reduction. Followed by one modified Inception/ResNet Block-A and Reduction Block-A. Terminates with more complexity reduction and our standard label classifier. See Figure 4.4.
CNN with frozen Inception/ResNetV2 base	Contains an Inception/ResNetV2 feature extractor with frozen weights followed by two dense layer blocks (the additional, trainable feature layers and then our standard label classifier. See Figure 4.5.

Characteristics of model architectures used. Descriptions rely on the building blocks illustrated in Figure 4.2 for brevity. Input images are all resized to 80x80. All models terminate with a final dense layer and softmax activation.

remain reserved for evaluating model performance after all hyperparameter tuning has been completed.

4.6.2 Phase II – Build and Train Source-Domain-Only CNNs

Three, non-DANN CNN architectures will be built, trained, and tuned during this phase on the satellite clips only. More thorough model descriptions are presented in Table 4.3 and are illustrated in Figures 4.3, 4.4, and 4.5. Of particular importance is the model feature extraction block in each of the models. These feature extractor architectures will be used as-is for Phase III and within the DANNs created, tested, and evaluated in Phase V.

As is standard, training will be completed using the training data and tuning completed based on the resultant model’s evaluation against the validation set. Both the 120x120

and 160x160 datasets will be evaluated. Clips from both of these datasets will be resized to 80x80 upon input into the models for training. This is intended as a meet-in-the-middle compromise for future evaluation of these models on the 40x40 target-domain UAS video clips. Decreasing the size of the satellite clips will result in a loss of information, whereas increasing the UAS clips may result in introduction of “new” information which is undesirable.

4.6.3 Phase III – Build and Train Target-Domain-Only CNNs

Phase III will be completed just as in Phase II, but using the target domain dataset only. Architectures evaluated will also remain the same with no hyperparameter tuning. Model performance will be evaluated against the UAS 40x40 clip size dataset. As in Phase II, model input size will be 80x80.

4.6.4 Phase IV – Evaluate CNNs on Source and Target Test Data

In Phase IV, the empirical bounds for DANN performance will be determined. As described in 4.3, the target domain (aerial) test dataset will be evaluated on non-DANN source-only trained CNNs and a target-only trained CNNs. The results of these tests will provide the lower and upper bounds of performance for the DANNs to be trained in Phase V.

In this phase we will continue to evaluate performance differences between the 120x120, 160x160, and 40x40 clip sizes. The target input size for each model will be 80x80.

4.6.5 Phase V – Build, Train, and Evaluate DANNs

In Phase V, the actual performance of DANNs will be determined. The two most performant CNNs from Phase IV will be modified to include a domain classifier with its GRL. These DANNs will be trained using the source-domain training data and labels to train the label classifier. Both the source-domain training data and the target-domain training data will be used to train the domain classifier. Explained further in 4.7, labels for the domain classifier will be generated for training with source-domain data labeled as “[1,0]” and target-domain data labeled as “[0,1].”

For evaluating performance of the DANNs during training, and for hyper-parameter tuning, the validation sets will be used. The label classifier will be evaluated against the target-

domain validation set and labels. The domain classifier will be evaluated against both the source- and target-domain data and generated labels (again “[1,0]” for source “[0,1]” for target). As in previous phases, the various combinations of generated clip sizes will be used for training to determine optimal size and mixing appropriate for overcoming the resolution disparity between domains.

Additionally, slight modifications to previous phase CNNs will be required to make them suitable for use in a DANN. For instance, consider the use of a frozen feature classifier, such as the Inception/ResnetV2 base, combined with a new label classifier and domain classifier. If this base is to remain frozen, then the gradients backpropagating through the network will be unable to modify the weights of the frozen feature classifier to become domain invariant. To overcome this, additional layers must be placed between the frozen base and label-/domain- classifiers, or the base must be unfrozen. Unfreezing the base however will introduce a substantial requirement for additional compute power and time.

The final goal of this phase, and research, will be to provide insights into the data prepossessing, performance, and architectural considerations for the use of DANNs in such an application.

4.7 Data Analysis and Measurements of Performance

All data labels have been structured as one-hot representations of the data’s corresponding class. This is done, even for the binary notRoad/road ([1,0], [0,1]) and source/target-domain ([1,0], [0,1]) classifiers to remain consistent with the CNNs used for [3]. To support this, all CNN architectures and their classifiers (whether label or domain) for this research will terminate in a hidden layer followed by a softmax activation function. This output provides the probability that each input belongs to each of the possible output categories.

For determining the accuracy of each classifier during every phase of research, the output of the softmax activation will be fed into the TensorFlow CategoricalAccuracy() method. This method compares the highest value output from the model to the one-hot truth values for each input. If the category with the highest output value matches the one in the on-hot truth values, CategoricalAccuracy() will record a correct answer, otherwise nothing. The final result over the data input is then divided by the total number of inputs for essentially an average or percentage of correctly labeled features.

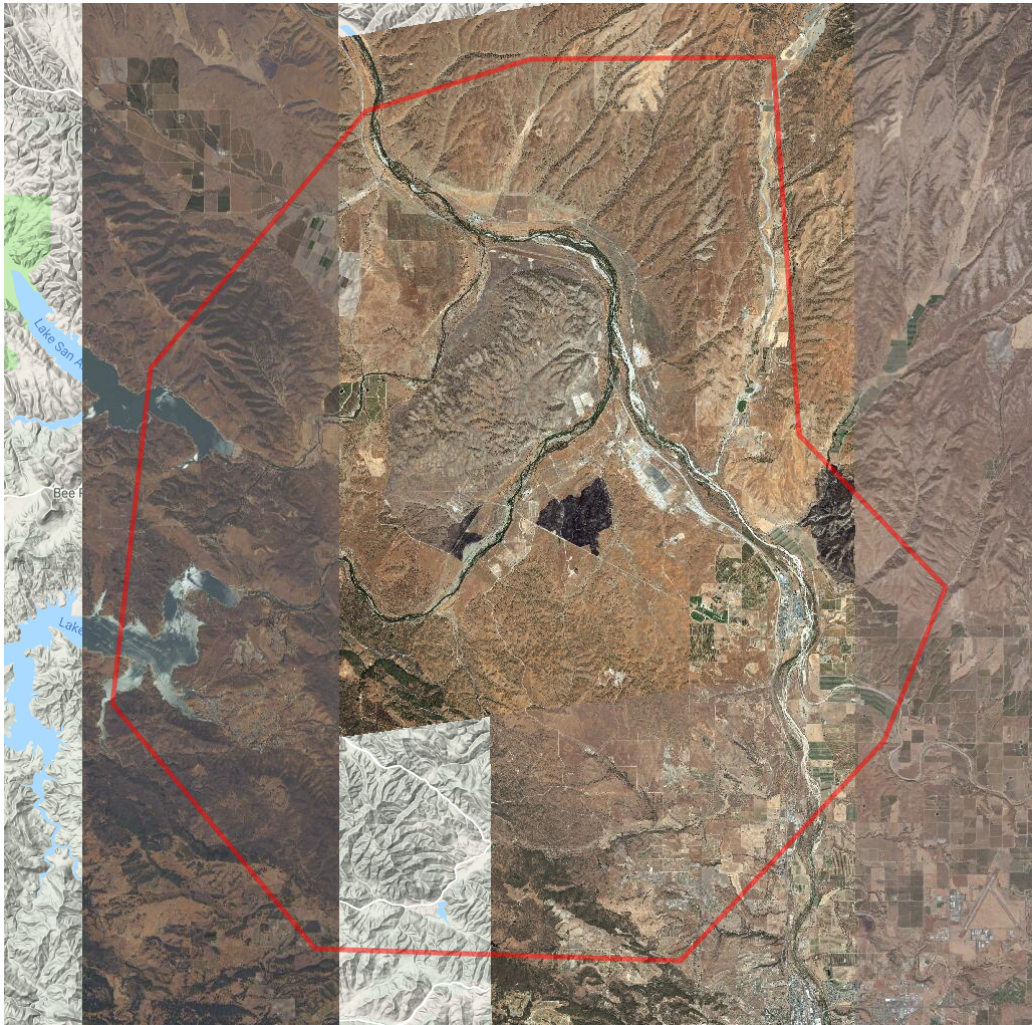
As discussed previously, we intend to determine the lower and upper theoretical bounds of accuracy for any DANN trained on the data. The lower bound essentially represents attempting to classify data from a dissimilar domain without any DA efforts (in other words, if we did not do anything). The upper limit represents the potential gain by trying DA on a model. The distance between the lower and upper bounds can be seen as the value-space for using a DANN. If this distance is small, then perhaps using a DA may not be a worthwhile solution for a particular application. If this distance is large, then DA should be looked at (unless resulting from domains that are far too dissimilar).

The effectiveness of DANNs for our particular application will be measured by the distance closed between our empirical lower and upper bounds towards the upper bound. If this improvement is marginal regardless of architecture, then we can know that DANNs may not be the best form of DA for adapting satellite trained CNNs to unlabeled UAS data.

4.8 Summary

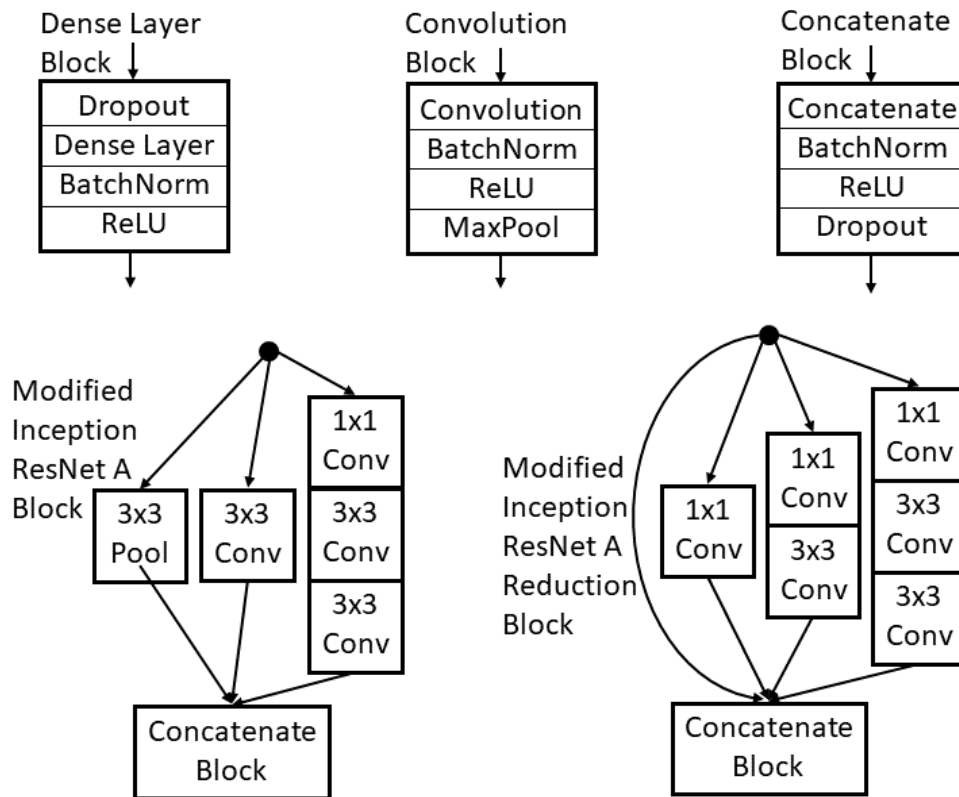
DANN evaluation has been broken into five phases meant to identify lower and upper bounds and actual DANN performance on several different input datasets and model architectures. These various datasets are to be created from from overhead imagery of Camp Roberts as satellite clips of 120x120 and 160x160 pixels and UAS clips of 40x40 pixels. Architectures to be used include a simple 6-layer CNN; a small, custom Inception/ResnetV2 inspired CNN; and a CNN using the frozen base (feature extractor) of a trained Inception/ResNetV2 model. Sources of data, hardware, and software used have been discussed in detail throughout this chapter.

Figure 4.1. Camp Roberts region and satellite collection area.



An overview of the Camp Roberts area from which satellite data was collected. The several distinct satellite collection passes that composed the image mosaic are visible. Clips for the source dataset were pulled from the red-bounded region.

Figure 4.2. Base model architectural building blocks.



These are common intermediate level NN blocks created for our custom models. These blocks incorporate NN improvements developed since [3] was published. They are used frequently within the three model architectures to be used for this research as illustrated later in this chapter.

Figure 4.3. Simple CNN architecture.

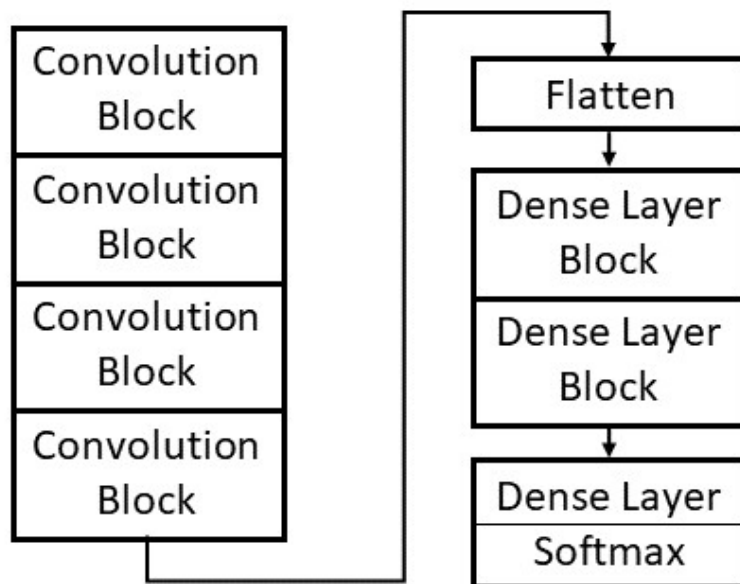


Figure 4.4. Small Inception/ResNetV2-ish CNN architecture.

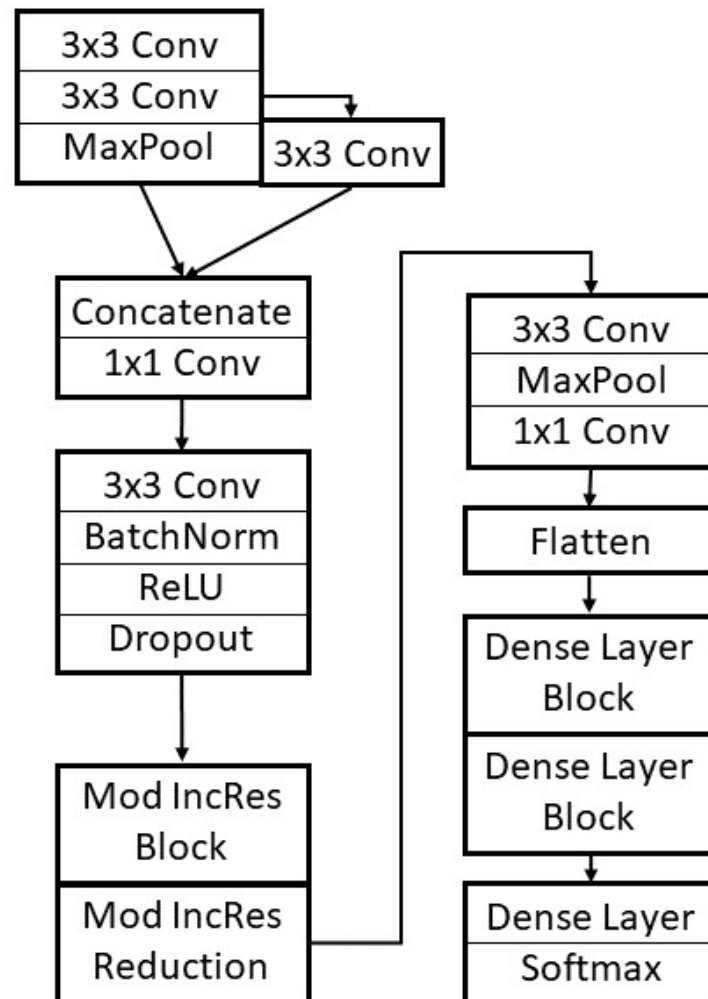
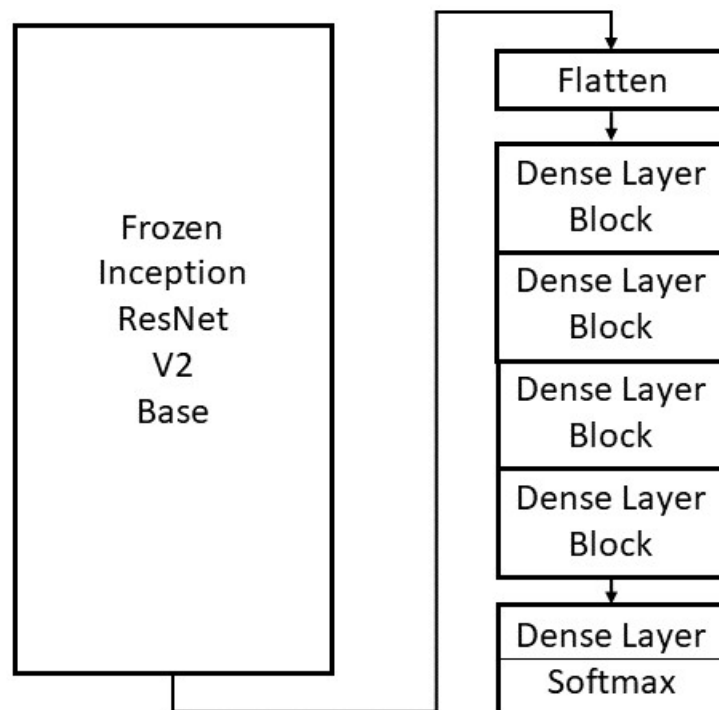


Figure 4.5. CNN architecture using frozen Inception/ResNetV2 base.



CHAPTER 5:

Results

5.1 Introduction

Research and experimentation for this thesis has sought to evaluate the benefits of DA, implemented through DANNs, in the cross-domain identification of roads between satellite imagery and UAS video. The results of this work will be presented throughout this chapter according to the phases of experimentation outlined in Chapter 4. Analysis of these results, identified limitations, and opportunities for future research will be discussed in Chapter 6.

5.2 Phase I – Define Domains and Build Datasets

Domains were defined while writing Section 4.6.1 prior to starting the labeling process. Lessons learned, regarding the weaknesses of these domain definitions, are discussed in Section 6.3.

This phase of the research was easily one of the most tedious. Manually creating the road vector data through ArcGIS took over 50 hours (though it could have been less if kept more localized to where UAS video had been acquired). Manually extracting frames then creating and labeling the 40x40 clips from video took over 36 hours. The 86 total hours of staring at a computer screen generating labels to facilitate ML for this application is a strong validating argument in favor of finding a robust DA tool. The ability to use pre-existing labeled datasets to generate models for unlabeled data would be a huge time (neck and back) saver.

At the conclusion of this phase, several labeled datasets had been created as summarized in Tables 5.1 and 5.2. Two satellite clip sets were made with clip size 120x120 and 160x160 with strides of 60 and 80 respectively. After filtering (described in Section 4.6), the size of these datasets remained substantial. One UAS video clip set was manually created with clip size 40x40.

Table 5.1. Satellite imagery datasets

Dimensions	Stride	% Rd/NRd	Train	Validate	Test	Processing Time
160x160	80	37.7 / 62.4	168576	84288	84289	96h 15m
120x120	60	30.5 / 69.5	299419	149710	149710	213h 07m

Distribution of generated satellite datasets. A sampling from the 160x160 “road” class is shown in 5.1.

Table 5.2. UAS video dataset

Dimensions	Stride	% Rd/NRd	Train	Validate	Test	Processing Time
40x40	20	33.0 / 67.0	60720	30361	30361	9h 09m

Distribution of generated UAS video datasets.

5.3 Phase II – Build and Train Source-domain-only CNNs

During this phase, three model architectures were trained and tuned using the training and validation source-domain only datasets. All three model architectures evaluated outperformed the source-only model used in [3] for the MNIST datasets. This is as expected given the simple structure of that model. Additionally, all three models performed well on the 120x120 and 160x160 datasets after initial adjustments to structure and associated hyperparameters. The highest accuracy achieved for each model/dataset is presented in Table 5.3.

The Adam optimizer provided the best results in all cases. Batch sizes were set to 64 with a learning rate of 1×10^{-6} and momentum set to 0.9. Dropout was set to 0.5 and L2 regularization was introduced and set to 0.2. Early stopping was used to prevent unnecessary training time in most cases, though set at 50 due to a mid-training plateau for the custom CNN architecture. Training was extended only when it was clear that training stopped too

Table 5.3. Source-domain only CNN Accuracy

Architecture	Dataset	Train	Validate	Time	Stop Epoch
Simple 5-layer	MNIST	98.40%	99.08%	5h 23m	600
	120x120	92.47%	91.14%	116h 38m*	600
	160x160	89.37%	91.06%	29h 06m	600
Custom Inc/ResV2-like	MNIST	98.48%	97.25%	4h 26m	248
	120x120	82.78%	84.54%	62h 24m	457
	160x160	83.52%	81.64%	59h 05m	600
Frozen Inc/ResV2 Base	MNIST	98.96%	99.92%	37h 06m	277
	120x120				
	160x160	99.88%	90.00%	291h 08m	600

Maximum accuracy of the three evaluated CNN architectures during training and validation and the total number of epochs after early stopping or after the maximum set 600 epochs was reached. A “*” denotes that the model was not trained on the machine described in Chapter 4.

early: i.e., a clear upward, but oscillating, trend in performance. An absolute maximum of 600 epochs was used.

5.4 Phase III – Build and Train Target-Domain-Only CNNs

Phase III was essentially a repeat of Phase II, but using the target-domain training and validation datasets instead. Hyperparameters from Phase II were used without modification. The highest accuracy achieved for each model/dataset is presented in Table 5.4.

As in phase II, all three model architectures evaluated outperformed the target-only model used in [3] for the MNIST-M datasets. This was also expected given the simple structure of that model. Additionally, all three models performed well on the 40x40 dataset. This

Table 5.4. Target-domain only CNN Accuracy

Architecture	Dataset	Train	Validate	Time	Stop Epoch
Simple 5-layer	MNIST-M	98.14%	96.28%	8h 47m	600
	40x40	91.23%	87.50%	10h 05m	600
Custom Inc/ResV2-like	MNIST-M	93.72%	95.78%	5h 31m	299
	40x40	83.16%	83.98%	7h 13m	269
Frozen Inc/ResV2 Base	MNIST-M	99.94%	94.48%	98h 58m	415
	40x40	99.66%	86.50%	105h 56m	600

Accuracy of the three evaluated CNN architectures during training and validation.

was not expected; it was presumed that information lost due to the lower resolution inputs would end up with lower performance. Results on the MNIST-M data was better than that in [3].

5.5 Phase IV – Evaluate CNNs on Source and Target Domain Test Data

Phase IV established the lower and upper bounds for the performance of Phase V DANNs. All trained models were re-evaluated against both source and target domain test datasets. Results are presented in Tables 5.5, 5.6, and 5.7. The performance of models trained on the 40x40 UAS video data is included in those tables.

5.6 Phase V – Build, Train, and Evaluate DANNs

The most performant architectures, training time considered, were the simple CNN and custom CNN architectures. Due to this, the DANN versions of these were used for the satellite/UAS DA task. The transfer learning models proved so complex that they took more time than administratively available for this research. To familiarize ourselves with DANN

Table 5.5. Evaluation on MNIST/MNIST-M CNN

Architecture	Trained on	Tested on	Accuracy	Evaluation Time
Simple 5-layer	MNIST	MNIST	98.32%	0m 02s
	MNIST	MNIST-M	32.70%	0m 03s
	MNIST-M	MNIST-M	96.40%	0m 03s
Custom Inc/ResV2-like	MNIST	MNIST	98.58%	0m 18s
	MNIST	MNIST-M	57.13%	0m 18s
	MNIST-M	MNIST-M	95.80%	0m 20s
Frozen Inc/ResV2 Base	MNIST	MNIST	98.49%	0m 32s
	MNIST	MNIST-M	18.90%	1m 00s
	MNIST-M	MNIST-M	94.47%	0m 32s

Accuracy of the highest performing models evaluated on the test sets. Source on source accuracy included for reference only.

performance while using transfer learning and the frozen Inception/ResNetV2 base, we did manage to train one MNIST/MNIST-M DANN.

During training for each of the DANNs listed in Tables 5.8, 5.9, and 5.10 a fair amount of oscillation in domain and label classifier accuracy occurred (to be discussed in Chapter 6). These DANNs were re-saved during training anytime a new validation accuracy max was achieved. These max-validation-accuracy DANNs were then evaluated against the MNIST-M and 40x40 test sets as identified in the tables below. Results from Phase IV are included in each of the tables to emphasize the relation of empirical lower and upper bounds as compared to the achieved accuracy from each DANN.

5.7 Summary

The evaluation of DANNs as a tool for cross-domain labeling of UAS video has shown they do provide modest improvements in labeling accuracy relative to non-DA CNN. This

Table 5.6. Evaluation on 120x120 trained CNN

Architecture	Trained on	Tested on	Accuracy	Evaluation Time
Simple 5-layer	120x120	120x120	91.08%	1m 56s
	120x120	40x40	50.66%	0m 08s
	40x40	40x40	87.40%	0m 17s
Custom Inc/ResV2-like	120x120	120x120	84.55%	5m 43s
	120x120	40x40	63.26%	1m 02s
	40x40	40x40	83.99%	1m 02s
Frozen Inc/ResV2 Base	120x120	120x120	84.55%	5m 43s
	120x120	40x40	63.26%	1m 02s
	40x40	40x40	86.35%	1m 38s

Accuracy of the highest performing models evaluated on the test sets. Source on source accuracy included for reference only.

this thesis also evaluated the use of more complex models and transfer learning for DANNs with positive results. Unfortunately, a number of limitations for DANNs has also been identified and will be discussed more in Chapter 6. Most notable of these limitations is a need for labeled target-domain data for validating the performance of trained models and/or clear metrics for when to end training of DANN when no labeled target-domain data is available. This requirement unfortunately invalidates their use for situations where an expectation of known labeling accuracy would be required.

Table 5.7. Evaluation on 160x160 trained CNN

Architecture	Trained on	Tested on	Accuracy	Evaluation Time
Simple 5-layer	160x160	160x160	89.47%	1m 26s
	160x160	40x40	53.71%	0m 22s
	40x40	40x40	87.40%	0m 17s
Custom Inc/ResV2-like	160x160	160x160	83.53%	3m 25s
	160x160	40x40	54.95%	1m 15s
	40x40	40x40	83.99%	1m 02s
Frozen Inc/ResV2 Base	160x160	160x160	90.04%	9m 19s
	160x160	40x40	67.81%	1m 39s
	40x40	40x40	86.35%	1m 38s

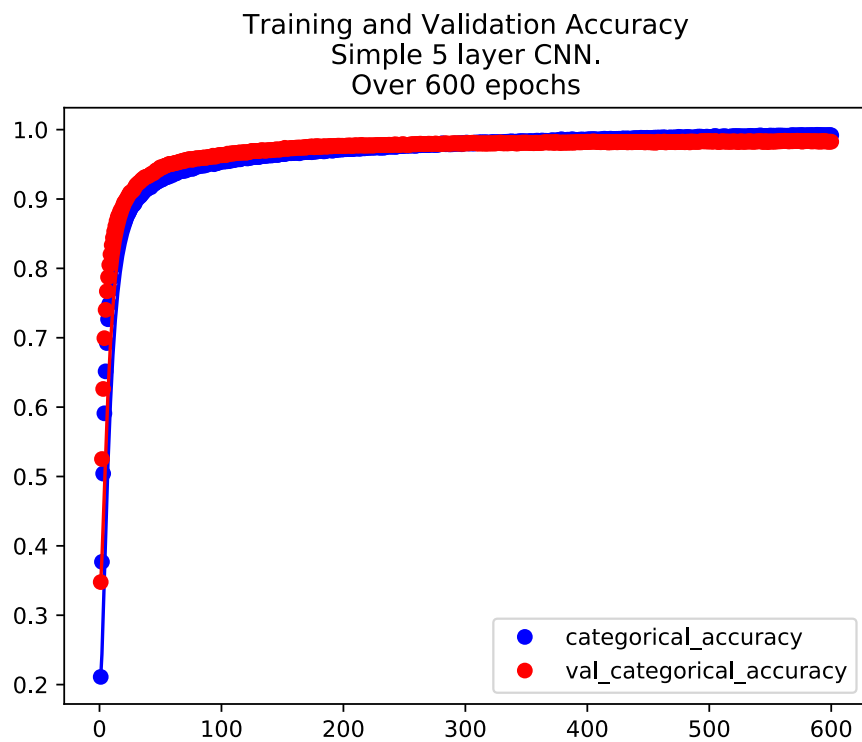
Accuracy of the highest performing models evaluated on the test sets. Source on source accuracy included for reference only.

Figure 5.1. Sample 160x160 “road” clips.



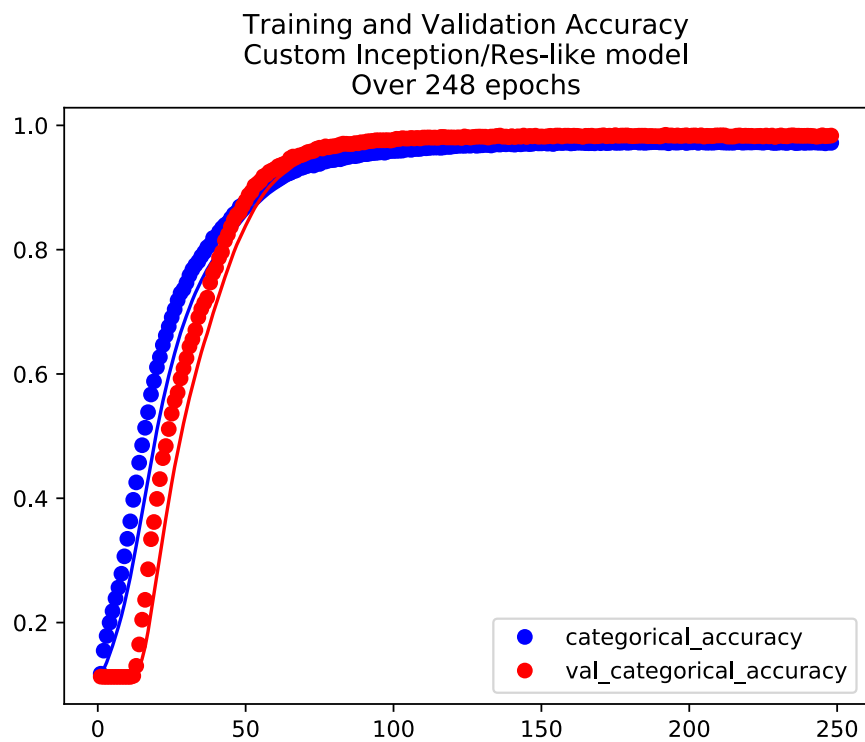
This is a sampling of the “road” class, as defined in 4.6.1 from the 160x160 satellite domain dataset. One can see how broad our domain definition for “road” actually was and how challenging some of these clips might be for the label classifier.

Figure 5.2. Training and validation for Simple CNN on MNIST data.



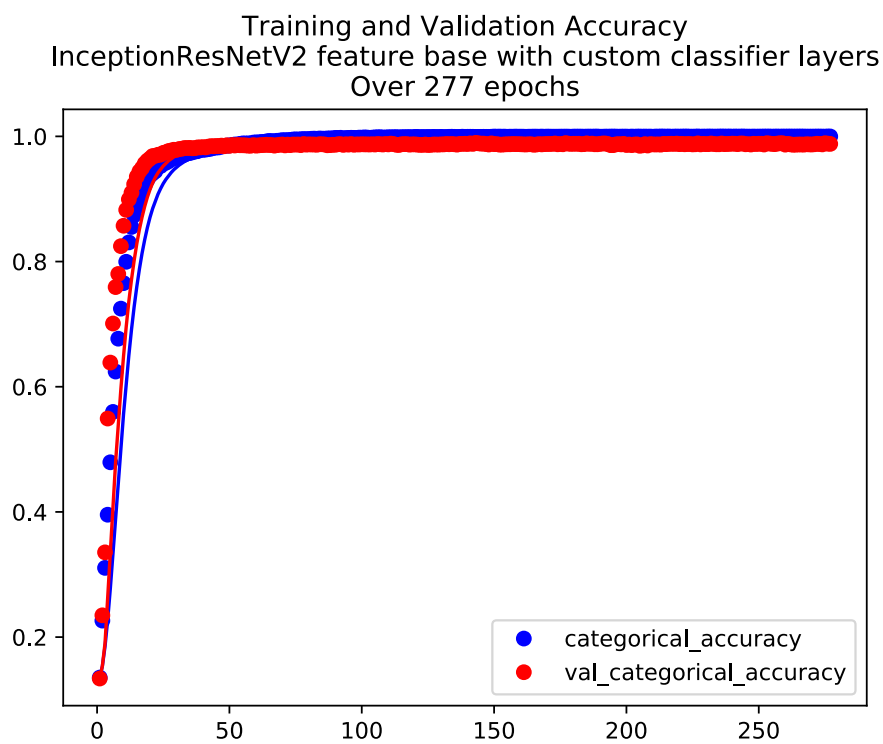
CNN training and validation plots for the “simple” architecture on MNIST data.

Figure 5.3. Training and validation for Custom CNN on MNIST data.



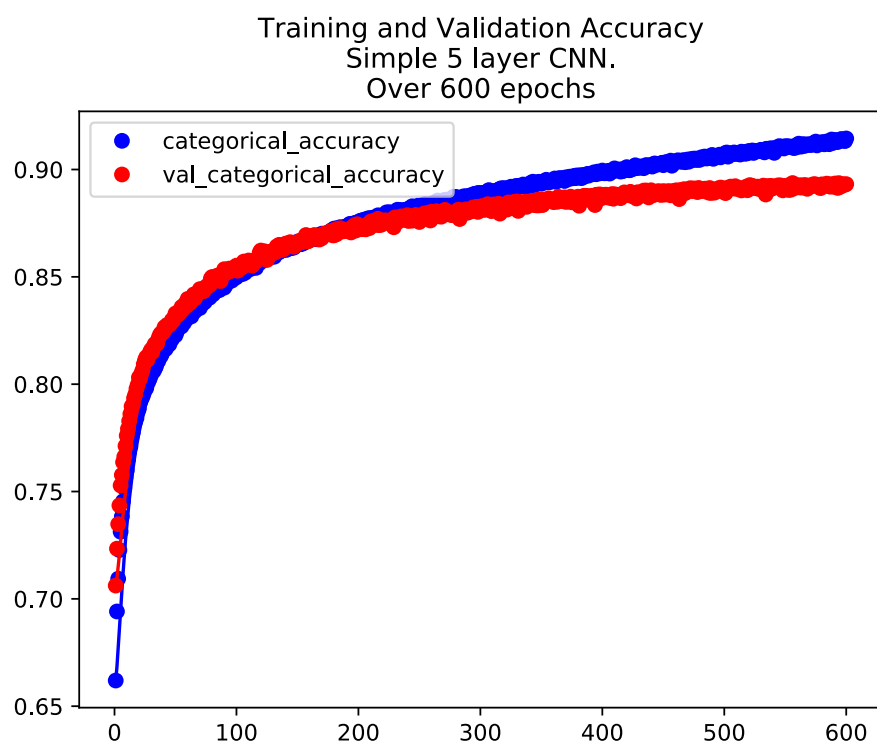
CNN training and validation plots for the “custom” architecture on MNIST data.

Figure 5.4. Training and validation for CNN with frozen feature extractor on MNIST data.



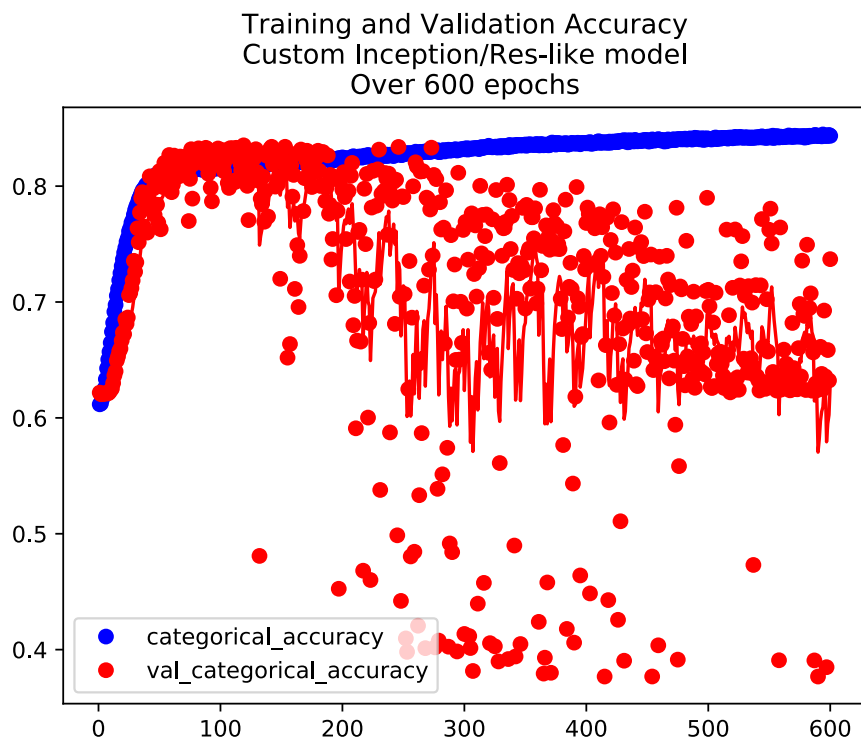
CNN training and validation plots for the “transfer learning” architecture on MNIST data.

Figure 5.5. Training and validation for Simple CNN on 160x160 satellite clips.



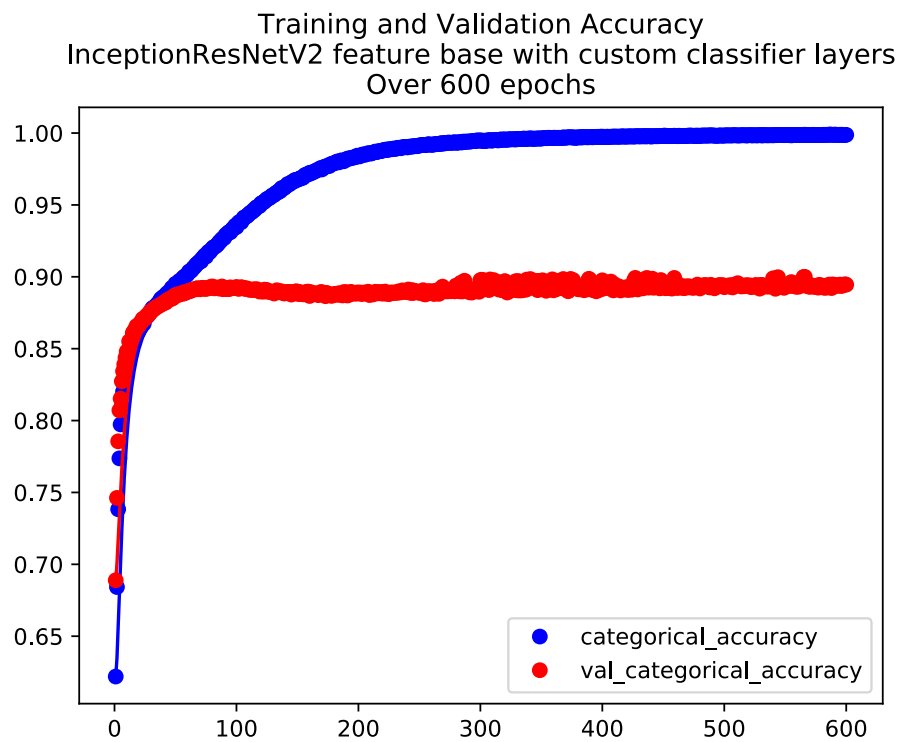
CNN training and validation plots for the “simple” architecture on the 160x160 satellite data.

Figure 5.6. Training and validation for Custom CNN on 160x160 satellite clips.



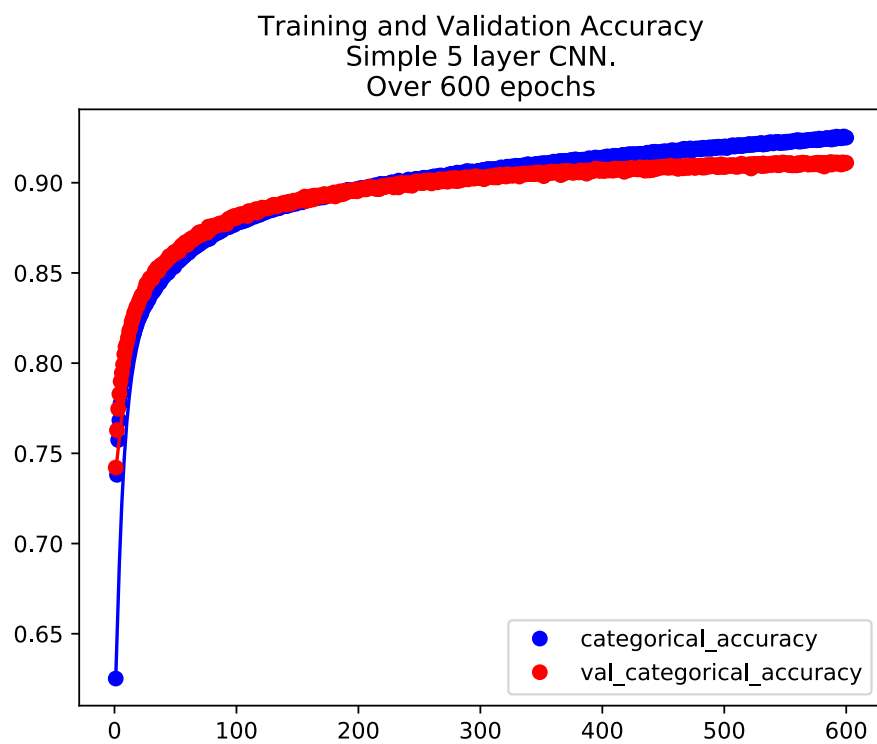
CNN training and validation plots for the “custom” architecture on the 160x160 satellite data.

Figure 5.7. Training and validation for CNN with frozen feature extractor on 160x160 satellite clips.



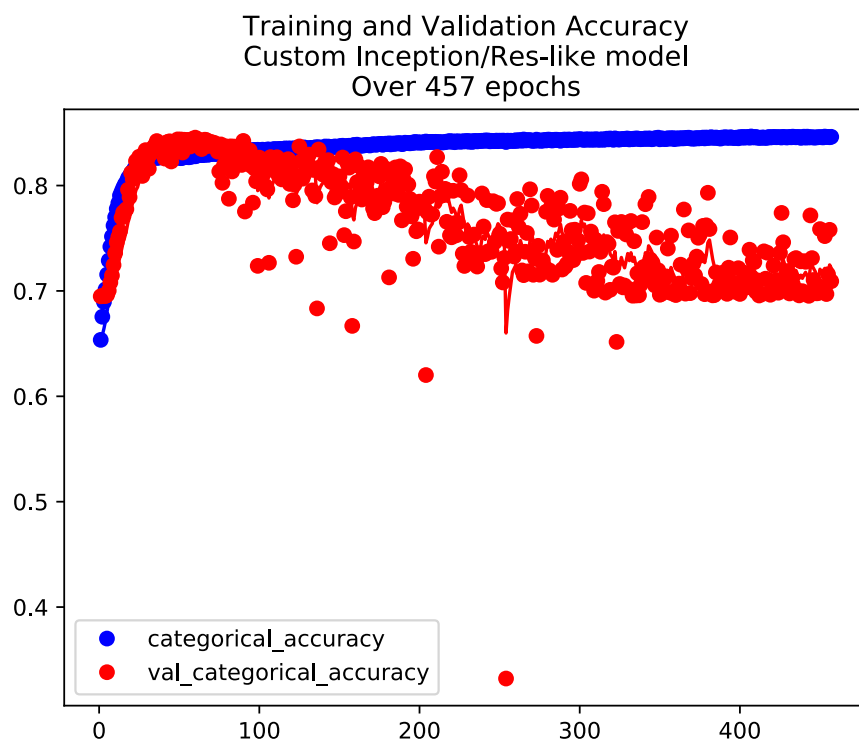
CNN training and validation plots for the “transfer learning” architecture on the 160x160 satellite data.

Figure 5.8. Training and validation for Simple CNN on 120x120 satellite clips.



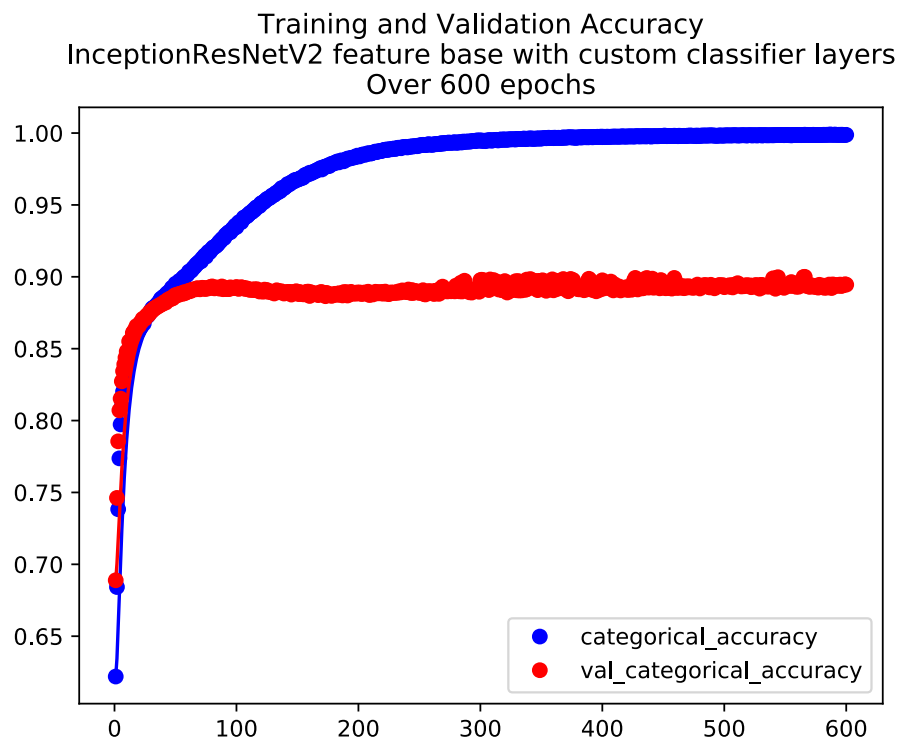
CNN training and validation plots for the “simple” architecture on the 120x120 satellite data.

Figure 5.9. Training and validation for Custom CNN on 120x120 satellite clips.



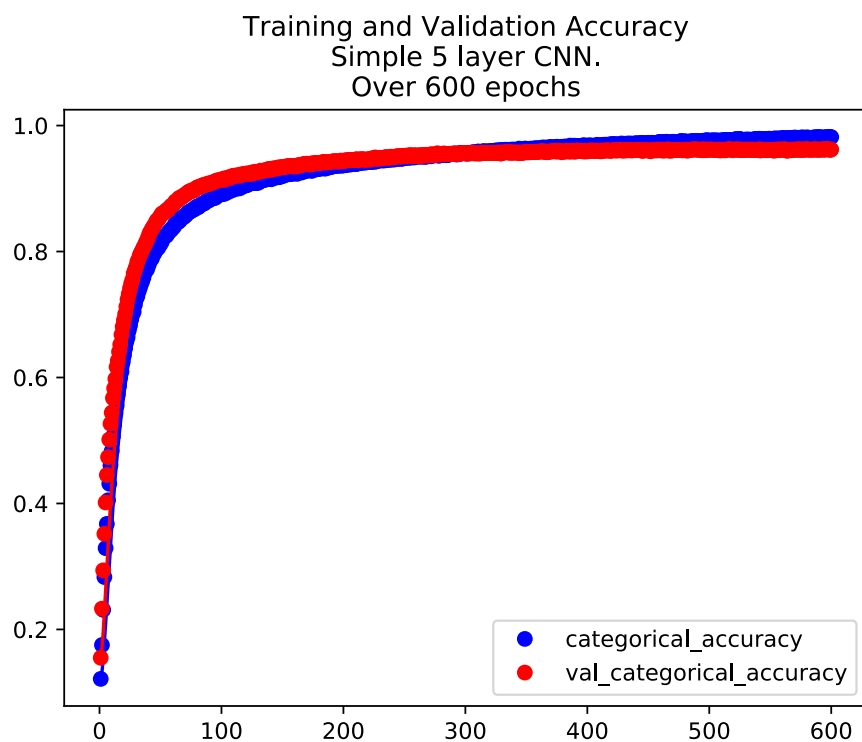
CNN training and validation plots for the “custom” architecture on the 120x120 satellite data.

Figure 5.10. Training and validation for CNN with frozen feature extractor on 120x120 satellite clips.



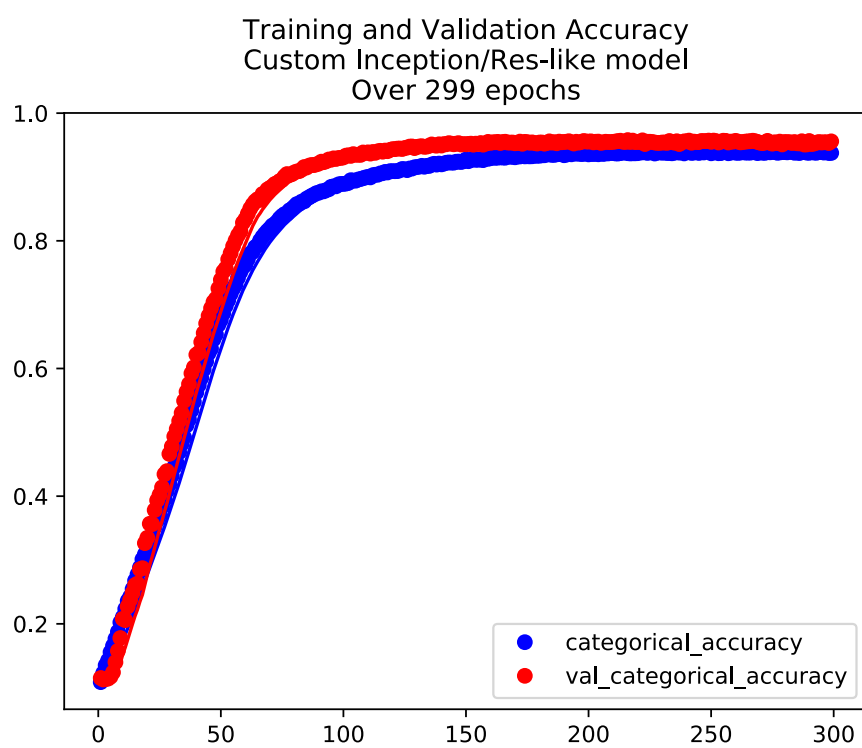
CNN training and validation plots for the “transfer learning” architecture on the 120x120 satellite data.

Figure 5.11. Training and validation for the Simple CNN on MNIST-M data.



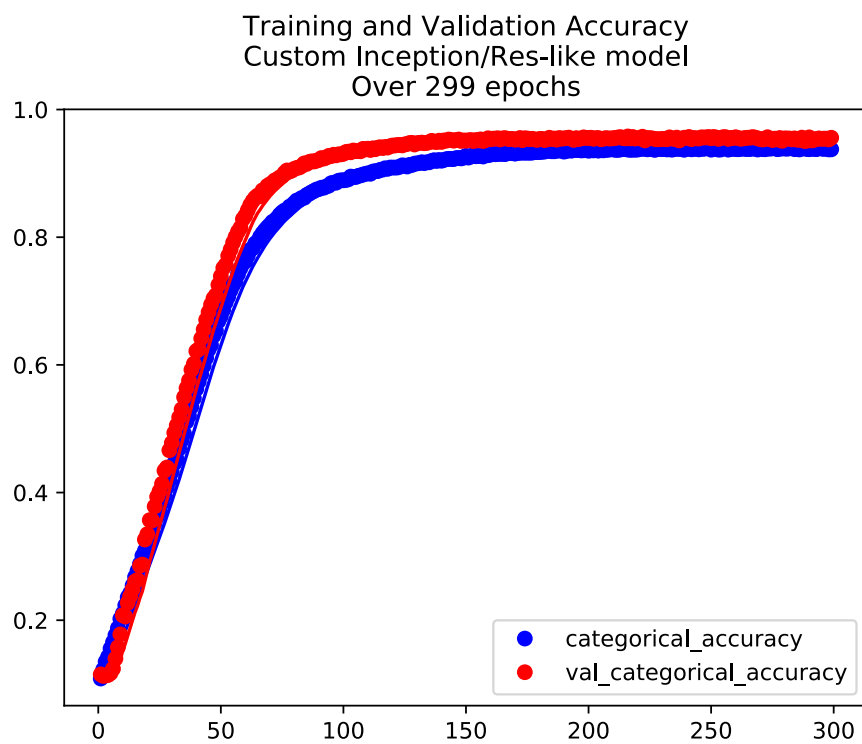
CNN training and validation plots for the “simple” architecture on MNIST-M data.

Figure 5.12. Training and validation for the Custom CNN on MNIST-M data.



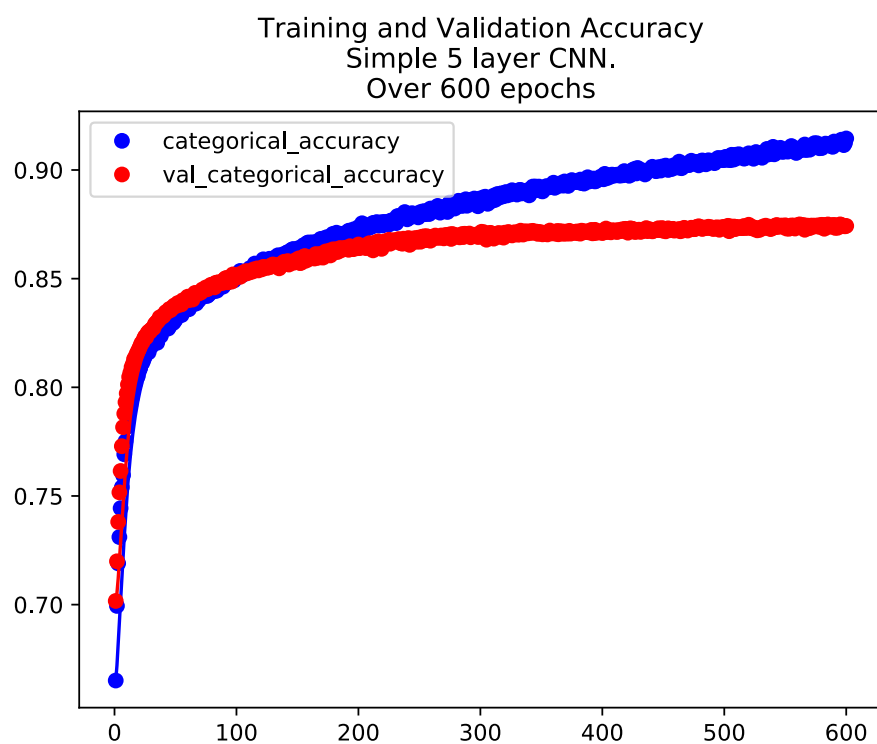
CNN training and validation plots for the “custom” architecture on MNIST-M data.

Figure 5.13. Training and validation for the CNN with frozen feature extractor on MNIST-M data.



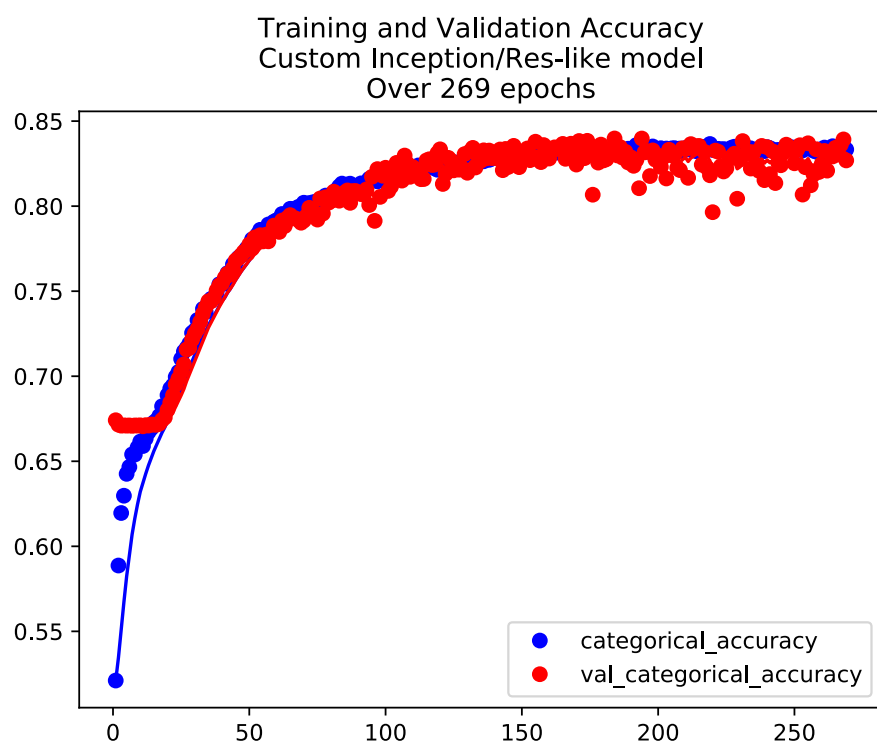
CNN training and validation plots for the “transfer learning” architecture on MNIST-M data.

Figure 5.14. Training and validation for the Simple CNN on 40X40 UAS video clips.



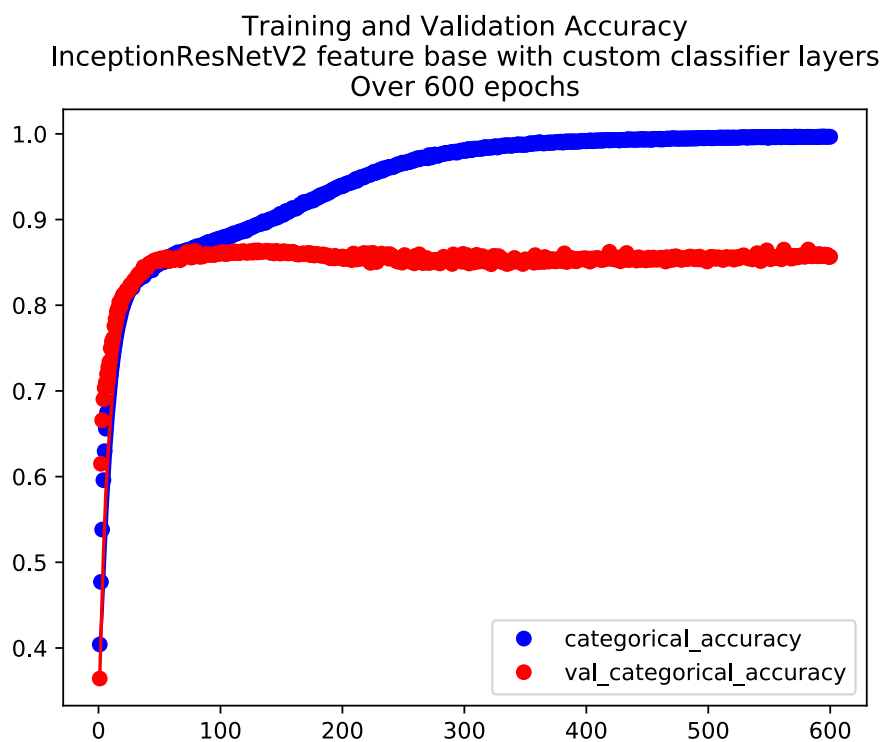
CNN training and validation plots for the “simple” architecture on the 40X40 UAS video data.

Figure 5.15. Training and validation for the Custom CNN on 40X40 UAS video clips.



CNN training and validation plots for the “custom” architecture on the 40X40 UAS video data.

Figure 5.16. Training and validation for the CNN with frozen feature extractor on 40X40 UAS video clips.



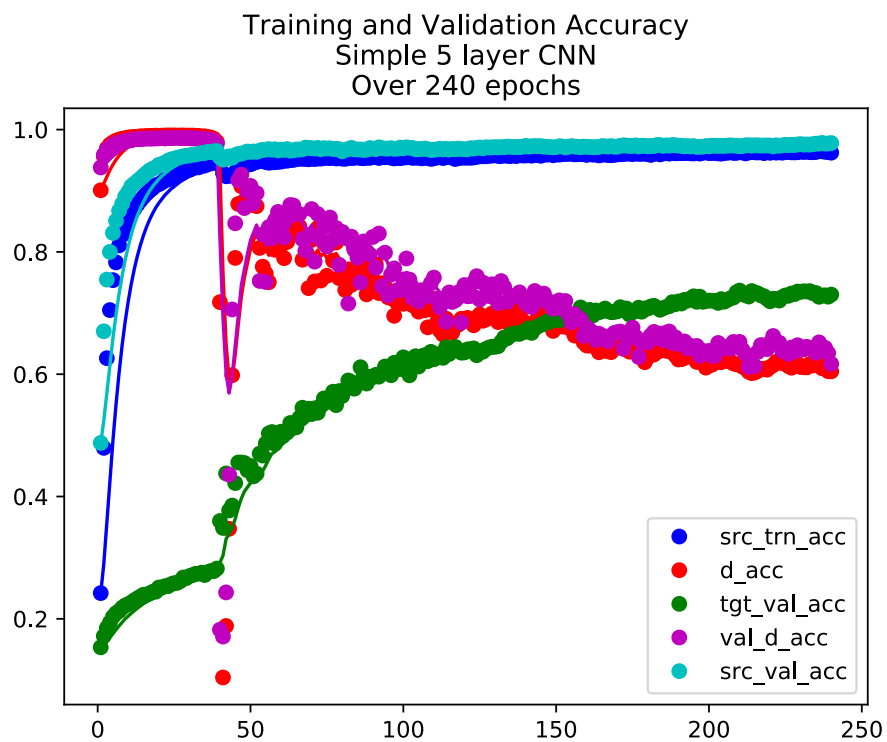
CNN training and validation plots for the “transfer learning” architecture on the 40X40 UAS video data.

Table 5.8. Evaluation of MNIST/MNIST-M trained DANN on MNIST-M test dataset

Architecture	Trained on	Tested on	Accuracy	Evaluation Time
Simple 5-layer	MNIST	MNIST-M	32.70%	0m 03s
	MNIST/MNIST-M	MNIST-M	74.33%	0m 03s
	MNIST/MNIST-M	MNIST	97.56%	0m 03s
	MNIST-M	MNIST-M	96.40%	0m 03s
Custom Inc/ResV2-like	MNIST	MNIST-M	57.13%	0m 18s
	MNIST/MNIST-M	MNIST-M	82.55%	0m 20s
	MNIST/MNIST-M	MNIST	98.56%	0m 20s
	MNIST-M	MNIST-M	95.80%	0m 20s
Frozen Inc/ResV2 Base	MNIST	MNIST-M	18.90%	1m 00s
	MNIST/MNIST-M	MNIST-M	59.96%	0m 27s
	MNIST/MNIST-M	MNIST	98.18%	0m 27s
	MNIST-M	MNIST-M	94.47%	0m 32s

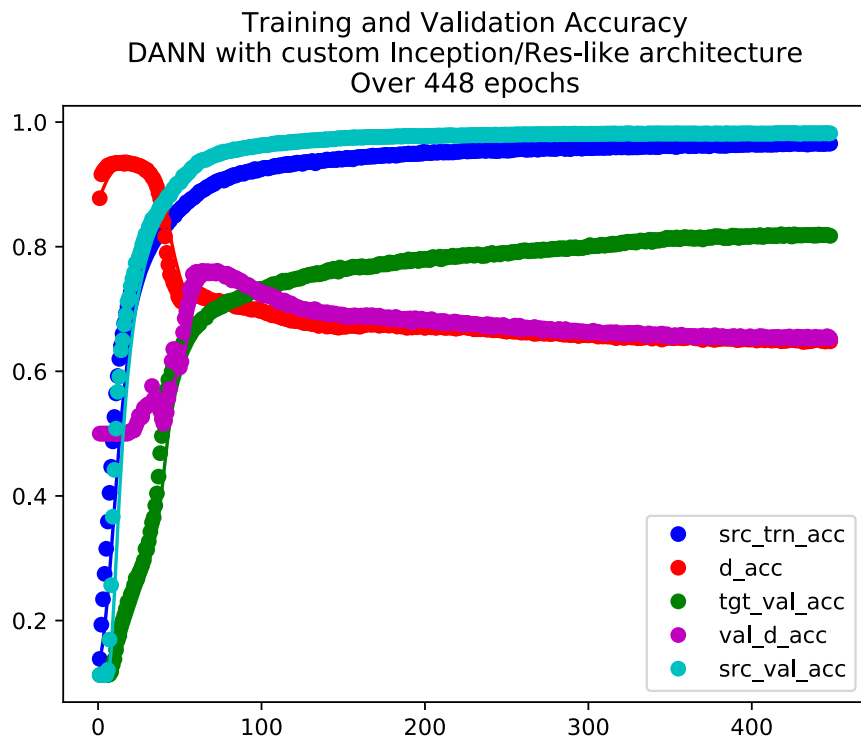
Accuracy of the highest performing architectures evaluated on the MNIST-M test set.

Figure 5.17. Training accuracy for Simple DANN on MNIST/MNIST-M data.



DANN plots for source training and validation accuracy, target validation accuracy, and domain training and validation accuracy for the “simple” DANN architecture on MNIST-M data.

Figure 5.18. Training accuracy for Custom DANN on MNIST/MNIST-M data.



DANN plots for source training and validation accuracy, target validation accuracy, and domain training and validation accuracy for the “custom” architecture on MNIST/MNIST-M data.

Figure 5.19. Training and validation for CNN with frozen feature extractor on MNIST/MNIST-M data.

DANN plots for source training and validation accuracy, target validation accuracy, and domain training and validation accuracy for the “transfer learning” architecture on MNIST/MNIST-M data.

Table 5.9. Evaluation of 120x120/40x40 trained DANNs on 40x40 test dataset

Architecture	Trained on	Tested on	Accuracy	Evaluation Time
Simple 5-layer	120x120	40x40	50.66%	0m 8s
	120x120/40x40	120x120	84.27%	2m 41s
	120x120/40x40	40x40	71.34%	0m 22s
	40x40	40x40	87.40%	0m 17s
Custom Inc/ResV2-like	120x120	40x40	63.26%	1m 2s
	120x120/40x40	120x120	81.32%	6m 52s
	120x120/40x40	40x40	69.40%	1m 15s
	40x40	40x40	83.99%	1m 2s

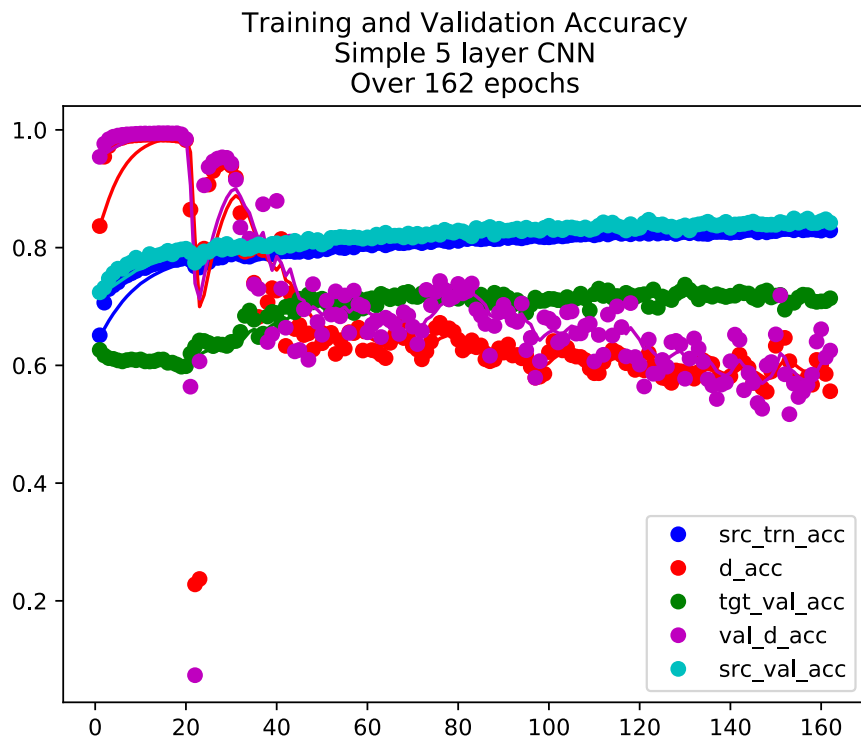
Accuracy of the highest performing architectures evaluated on the test sets.

Table 5.10. Evaluation of 160x160/40x40 trained DANNs on 40x40 test dataset

Architecture	Trained on	Tested on	Accuracy	Evaluation Time
Simple 5-layer	160x160	40x40	53.71%	22s
	160x160/40x40	160x160	81.59%	1m 19s
	160x160/40x40	40x40	65.28%	22s
	40x40	40x40	87.40%	0m 17s
Custom Inc/ResV2-like	160x160	40x40	54.95%	1m 15s
	160x160/40x40	160x160	80.93%	3m 41s
	160x160/40x40	40x40	66.64%	1m 9s
	40x40	40x40	83.99%	1m 2s

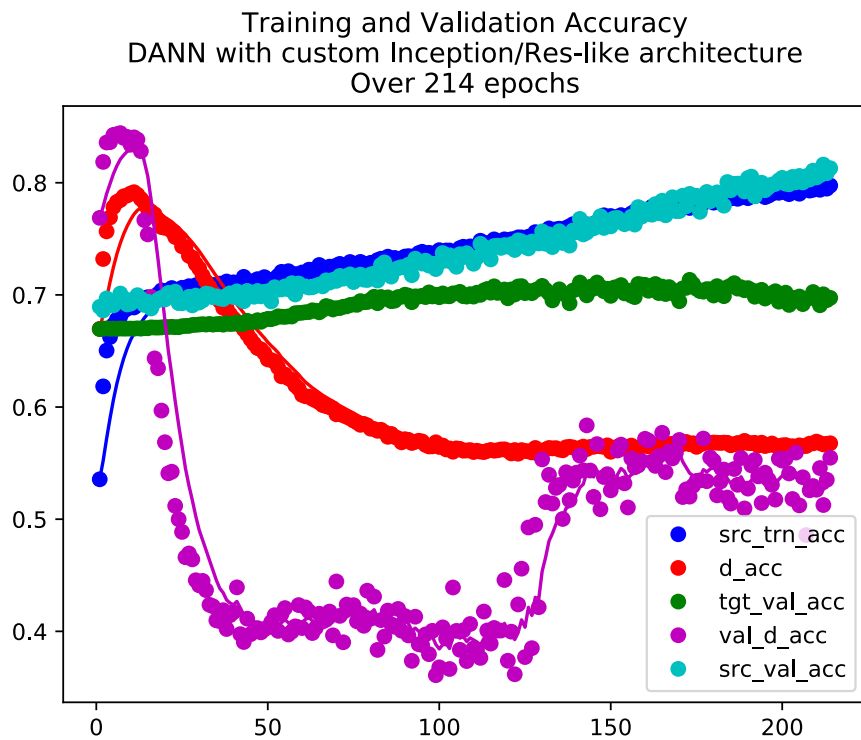
Accuracy of the highest performing architectures evaluated on the test sets.

Figure 5.20. Training accuracy for Simple DANN on 120x120/40x40 data.



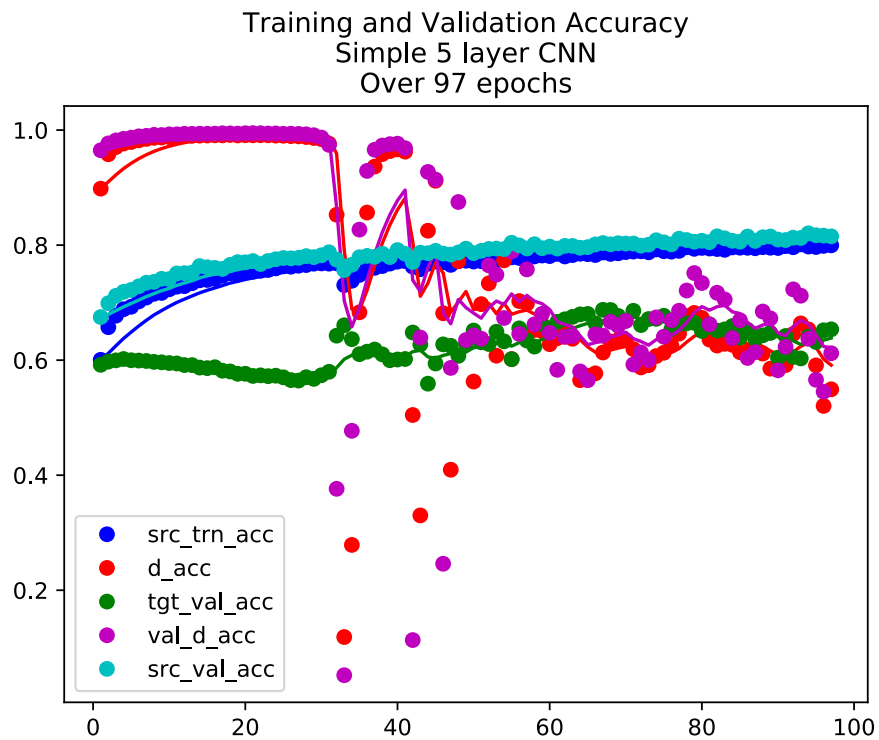
DANN plots for source training and validation accuracy, target validation accuracy, and domain training and validation accuracy for the “simple” DANN architecture on 120x120/40x40 data.

Figure 5.21. Training accuracy for Custom DANN on 120x120/40x40 data.



DANN plots for source training and validation accuracy, target validation accuracy, and domain training and validation accuracy for the “custom” architecture on 120x120/40x40 data.

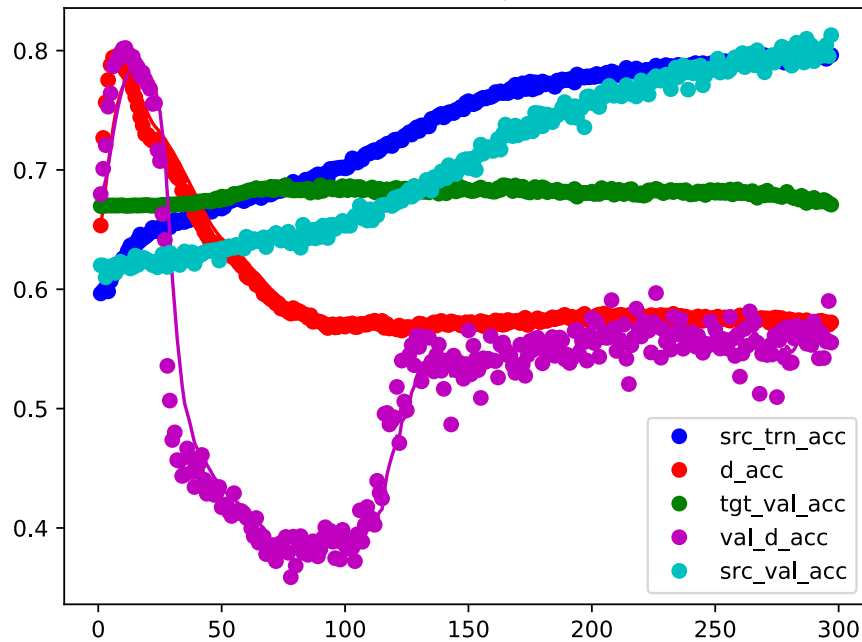
Figure 5.22. Training accuracy for Simple DANN on 160x160/40x40 data.



DANN plots for source training and validation accuracy, target validation accuracy, and domain training and validation accuracy for the “simple” DANN architecture on 160x160/40x40 data.

Figure 5.23. Training accuracy for Custom DANN on 160x160/40x40 data.

Training and Validation Accuracy
DANN with custom Inception/Res-like architecture
Over 297 epochs



DANN plots for source training and validation accuracy, target validation accuracy, and domain training and validation accuracy for the “custom” architecture on 160x160/40x40 data.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 6:

Conclusions, Limitations, and Future Work

6.1 Introduction

Following discussion of machine learning and remote sensing in Chapter 2, Chapter 3 delved into the specifics of DA and Domain Divergence, the theoretical underpinnings for DANNs. This provided context for our experimental design and results in Chapters 4 and 5. The purpose of these experiments was to determine if DANNs could provide a means for training a CNN with satellite data for use on UAS video without the need for labeled UAS video data. In this chapter, we pull everything together with the results of our experiments to discuss the implications of these results for our application, their limitations, as well as opportunities for future work.

We begin this chapter with an overall analysis of results from Chapter 5. Following this analysis, we combine discussion of specific limitations and open questions that remain from each phase of our research preparation and conduct.

6.2 Conclusions

At the outset of this research, we sought to determine whether or not a CNN trained to identify roads on satellite imagery could be adapted to identify roads from an aerial platform. As part of that process, we also sought an efficient way to create a large labeled dataset for this purpose and to determine the performance of CNNs for this task. We were interested in the unmodified performance of these CNN on dissimilar datasets and finally the resulting performance from applying the concepts discussed in [3] to our training of these CNNs.

Using existing tools, we were successful in developing a pipeline for generating large amounts of labeled satellite clips for our research. Developing this pipeline required overcoming several setbacks. The pre-existing road vector datasets initially intended for a fully automated process were not up-to-date and we experienced issue matching map projections across datasets. Manually creating our road vector dataset was time consuming,

but still substantially less involved and prone to error than manually labeling individual satellite clips (as experienced while manually labeling UAS clips).

Creation of the CNNs were relatively straightforward and took advantage of more recent NN research. For instance, while not specifically measured, anecdotal experience while training these models demonstrates a tremendous reduction in training time due to the use of Batch Normalization Layers. Dropout also performed well as a regularizer to help each of the models generalize better. Software support for building these models through TensorFlow 2.0 and its Keras high level API was robust and allowed for quick modifications to our model structures. This proved helpful while developing performant architectures for this application. This would have also proved helpful for conducting a more thorough hyperparameter tuning, had time allowed.

To ensure our models function correctly, we assessed our code and architectures on the same MNIST/MNIST-M datasets used for [3]. In all cases, as shown in Chapter 5, our CNNs outperformed the models published in [3], both in training time and classification accuracy. Our MNIST/MNIST-M DANN also showed marginal improvement, but in substantially fewer training steps. This demonstrates that more complex architectures, tailored to the classification task, can improve DANN performance. We also demonstrate that transfer learning and the architectural concepts used in the Inception and ResNet models can also improve DANN performance.

Furthering the work of [3], we have also shown that DANNs do provide an increase in cross-domain labelling accuracy in a remote sensing setting when target-domain labels are “unavailable.” Increases in performance, however, are modest at best; even when using architectures that capture more of the source and target domain complexity. We believe this modest level of improvement had more to do with curation of our datasets than the actual structure of the models used.

As a whole, we can say that a DANN modified CNN does help a CNN trained on satellite data to perform better on the cross-domain classification task on aerial video clips. Since our DANNs did not completely cross the gap from lower to upper empirical performance bounds another form of DA is worth exploration as are other architectures and data pre-processing techniques. The performance we experienced is likely due to a number of limitations that were identified during experimentation that would warrant further investigation (to be

discussed in Section 6.3).

An important observation is how well our DANNs performance adhered to the theory presented in [2], [3]. All trained DANNs performed within the lower and upper bounds identified in Phase IV of our work. These bounds are useful given the ambiguity to be expected in training a DANN without a labeled target dataset to validate performance against. These bounds are also useful while evaluating the type and form of data preparation or transformation that is best for training. In our work, we experienced the best DANN performance when using the 120x120 source satellite clips and 40x40 target UAS clips resized to 80x80 for input. This is expected as the 120x120 and 40x40 clips demonstrated the lowest domain divergence.

6.3 Limitations and Future Work

6.3.1 Domains and Labels

Domains were “well” defined in 4.6.1. During the labeling process, however, it became clear that they did not account for some of the more ambiguous scenes in the satellite and UAV video. For instance, a vehicle driving once through a field of tall grass would leave behind two parallel tracks of bent/broken grass, lighter than the surrounding grass that was not run over as seen in Figure 6.1. This single pass through the grass often appeared similar to apparently well worn trails. The minor difference in appearance (not captured by the domain definition of “Jeep Trail”), led to ambiguity in labeling (which would then transfer into any NN trained on such data). Paved and unpaved roads remained relatively true to their domain definitions.

Another notable source of ambiguity during the labeling process came in determining the difference between fire breaks and roads. Fire breaks are essentially one time use “roads” created to limit, slow, or control the spread of intentional control burns or out of control grass or brush fires. When they are newly created, they appear similar to other more off-the-beaten-path dirt roads. Since their purpose can be transient however, they can quickly fall into disrepair. At some point, the “road” vegetation overgrowth and water erosion gets to the point where the fire break is no longer usable as a road. This transition point is not easily discernible in remotely collected imagery.

Figure 6.1. Jeep trails in grass.



Imagery from the satellite dataset with vehicles driving through a grassy area.

Treed areas in the imagery often made it difficult to follow the exact route of roads and required a substantial amount of interpretation: looking for road entry/exit points in the canopy, breaks in the canopy, “linear” looking patterns relative to these breaks (things do not disappear just because you cannot see them). Collectively, all of this ambiguity and interpretation contributed to errors in the final dataset caused by the analyst producing the labels for training the models. Reducing this error in the data should translate into better results. Ground truth data would be most ideal, however it does not take much imagination to think of a scenario where satellite imagery exists for an area that ground truth data cannot be acquired.

A more specific breakdown of domains into multiple classes for this application would be worthy area of exploration. DANNs have shown utility for the multi-class MNIST problem, so could also have utility for a DANN that could distinguish between more than just “road” or “not-road.”

6.3.2 Information Loss

Resolution

A very notable source of error and ambiguity came simply from the tremendous amount of “information” loss between the 0.5 meter resolution satellite imagery and the 2 meter UAV video. In one dimension, the video data contained only $\frac{1}{4}$ the information available in the satellite imagery. In both dimensions, this equated to a 16 times reduction. For example, consider a feature on the ground that was 2x2 meters in size. In the satellite imagery, this feature would take up at least 16 pixels. In the UAV video, this feature would appear as just one pixel, or may even be blurred across four pixels (squared). The simple ability to “see” a feature is a requirement for an analyst, or CNN in this case, to identify it.

In this scenario, with the datasets in use, the Jeep Trails and Game/Foot Trails were not discernible in the UAS datasets. This is due to a width generally less than 1 meter and a calculated UAS GSD of 2 meters. None of these trails could ever fill enough of a pixel to be detectable. This disparity could reasonably be expected to further limit our CNN and DANN performance, particularly when the satellite datasets did have discernible trails (by increasing domain divergence). Due to administrative time constraints and the length of time required for training our NNs, we were not able to create new, labeled datasets which accounted for this difference.

To standardize clip input, the satellite clips and UAS clips were all resized to 80x80. This changed the original “GSD” for each domain further. Changing the 40x40 clips to 80x80 essentially doubled its “resolution” from 2 meters to 1 meter (and approximated information to do so). Changing the 160x160 clips halved their resolution to approximately 1 meter and changing the 120x120 clips reduced their resolution by a third to approximately 0.75 meter (and removed information to do so). The DANN trained using the 120x120 and 40x40 clips performed best, perhaps because there was less manipulation of the data prior to training. Evaluating the range of inputs and their transformations to a DANN for this use case is

warranted.

Clips without Context

While creating the satellite road vectors and the UAS clips, it became apparent that the entire scene (including areas outside a labeled clip) was helpful in identifying and labeling roads. For instance, a linear feature may originate and terminate between two road intersections, this information helps validate this feature as a road. Such spatial information was discarded when each clip was extracted from the larger scene and the surrounding context was not saved. Since it was so useful to have this information, it would likely also be helpful to a NN structured to take advantage of it. Further iterations of this research should look into the performance boost from maintaining the spatial context in which each clip is generated from.

6.3.3 Multiprocessing

One of the most time consuming parts of processing this data was due to a lack of parallel processing capability built into ArcGIS's `ExportTrainingDataForDeepLearning()`. The consequences of this were substantial. Leaving 11 of 12 CPU cores idle, one run of `ExportTrainingDataForDeepLearning()` took nearly nine days. Fortunately, the way this tool was structured to access data, we were able to run several instances of `ExportTrainingDataForDeepLearning()` simultaneously. As awesome as this tool was, it would be a major improvement in the process if this tool were reprogrammed to take advantage of multiprocessing.

While developing code for this thesis, we prioritized maximizing use of available system resources through Python's Multiprocessing package where able. The improvement in performance was substantial. Consider, for instance, our code for filtering and sorting clips output by the ArcGIS `ExportTrainingDataForDeepLearning()` method. Initial implementation of this filter did not use multiprocessing and took typically 15 minutes to complete per run. Once multiprocessing was implemented, this step of our pipeline was reduced to 1.5 minutes. The ability to quickly iterate over the data and to correct issues provides a tremendous advantage when evaluating optimal characteristics and combinations of data input to our models.

6.3.4 DANN Training and Performance Characteristics

Hyperparameter Search and Architectural Tuning

Due to time constraints and experimentation with architecture and dataflow, a thorough hyperparameter search was not conducted. For optimal performance, such a review is important and warrants future work. While we experienced modest performance increases, its likely that tweaks to the hyperparameters used would facilitate even better cross-domain accuracy.

A critical component of cross-domain model performance, is a model’s ability to generalize across those domains. This could likely be improved through further modification to model architecture and regularization used in phases II and III. Generalization is also improved through use of more data with greater variety of examples. It would be interesting to see if generalization and performance increases with the use of more satellite data for the target area, adding satellite data from other areas, or training the models with artificially reduced resolutions or other forms of augmentation.

Our experimentation with transfer learning opened new questions for the structure of DANNs that use this concept. If the frozen feature base is included in the DANN architecture, then the DANN domain classifier is unable to adjust the weights of that base to be domain invariant. We identified two possible solutions for this: add layers after the frozen feature base or unfreeze the frozen base. Due to additional compute time expected from unfreezing the feature base, we opted for the addition of layers. More experimentation with this architecture is warranted as DANN performance was substantially lower than expected with a completely frozen base and extra layers.

Training Characteristics and Hyperparameter Scheduling

While training “normal” NNs, the relative performance of your model on training and validation datasets provide a fair indication of when model training should terminate. While training the DANNs, this was not the case. There were some very interesting oscillations in domain classifier accuracy on the training/validation datasets and on the label classifier accuracy on the target domain dataset. This is likely due to the adversarial relation between label and domain classifiers and the Adam optimizer used. Due to these oscillations, multiple adjustments to the threshold used for early stopping had to be made,

and in some instances ignored completely. Oscillations during training were not mentioned in [3], however no training performance graphs or discussion on training or tuning was present, so its unclear if their use of pure SGD for training will eliminate this phenomenon.

Presumably, DANNs will be used in situations where target domain labels are not available. Without the ability to validate performance on the target domain or a clear indication of when to stop training, one could easily end up with a sub-optimal DANN. Theoretically, their optimal performance should occur when the domain classifier performs at its worst (equivalent to guessing). During several test runs, however, the domain classifier blew through this threshold and temporarily provided better label performance by classifying each example incorrectly. If DANNs are to be used in the future, their training characteristics need to be better understood. This would be a worthwhile area for further exploration.

6.4 Summary

This research has shown that DA, combined with labeled satellite data, provides a viable solution to UAS CNN accuracy issues associated with a lack of labeled UAS video data. As the first to look at this issue, our research opens the door for quite a bit of future opportunities to evaluate this topic, particularly as we have validated its utility and identified a number of questions left unanswered in the work on which this research was based: [3]. We believe future work would show greater improvements to this DA task through better domain definitions and labeling for the source and target domains, use of UAS video with a higher GSD (resolution), integration of satellite/video clip context, better integration of multiprocessing, and a better study of DANN training and performance characteristics. A full solution for this task is not far off and will ultimately lead to true autonomy for the AI systems we are certainly going to rely on in the future.

List of References

- [1] S. Ben-David, J. Blitzer, K. Crammer, and F. Pereira, “Analysis of representations for domain adaptation,” in *Proceedings of the 19th International Conference on Neural Information Processing Systems (NIPS’06)*. Cambridge, MA, USA: MIT Press, 2006, pp. 137–144. [Online]. Available: <http://dl.acm.org/citation.cfm?id=2976456>. 2976474
- [2] S. Ben-David, J. Blitzer, K. Crammer, A. Kulesza, F. Pereira, and J. Vaughan, “A theory of learning from different domains,” *Machine Learning*, vol. 79, no. 1-2, pp. 151–175, May 2010.
- [3] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky, “Domain-adversarial training of neural networks,” *Journal of Machine Learning Research*, vol. 17, no. 1, p. 2096–2030, Jan. 2016.
- [4] F. Chollet, *Deep Learning with Python*, 1st ed. Greenwich, CT, USA: Manning Publications Co., 2017.
- [5] K. Lee, *AI Superpowers: China, Silicon Valley, and the New World Order*. New York, NY, USA: Houghton Mifflin Harcourt, 2018.
- [6] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning internal representations by error propagation,” in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*, D. E. Rumelhart, J. L. McClelland, and C. PDP Research Group, Eds. Cambridge, MA, USA: MIT Press, 1986, pp. 318–362. [Online]. Available: <http://dl.acm.org/citation.cfm?id=104279>. 104293
- [7] “Introduction,” class notes for CS231n: Convolutional Neural Networks for Visual Recognition, Dept. of Comp. Sci., Stanford University, Palo Alto, CA, USA, spring 2017. [Online]. Available: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture1.pdf
- [8] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2009.
- [9] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016. [Online]. Available: <http://www.deeplearningbook.org>
- [10] “Training neural networks, part i; neural nets notes-3,” class notes for CS231n: Convolutional Neural Networks for Visual Recognition, Dept. of Comp. Sci., Stanford University, Palo Alto, CA, USA, spring 2017. [Online]. Available: <http://cs231n.github.io/neural-networks-3/>

- [11] “Training neural networks, part i; neural nets notes-1,” class notes for CS231n: Convolutional Neural Networks for Visual Recognition, Dept. of Comp. Sci., Stanford University, Palo Alto, CA, USA, spring 2017. [Online]. Available: <http://cs231n.github.io/neural-networks-1/>
- [12] K. Willems. “Keras tutorial: Deep learning in python.” DataCamp. 2019. Accessed Jan. 10, 2020. [Online]. Available: <https://www.datacamp.com/community/tutorials/deep-learning-python>.
- [13] T. Renelle. “Deep learning.” Machine learning guide. 2017. [Online]. Available: <https://www.ocdevel.com/mlg/9>
- [14] G. B. Thomas, M. D. Weir, J. R. Hass, and F. R. Giordano, *Thomas’ Calculus Early Transcendentals (11th Edition) (Thomas Series)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2005.
- [15] “Loss functions and optimization,” class notes for CS231n: Convolutional Neural Networks for Visual Recognition, Dept. of Comp. Sci., Stanford University, Palo Alto, CA, USA, spring 2017. [Online]. Available: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture3.pdf
- [16] Y. Lecun, “Generalization and network design strategies,” in *Connectionism in perspective*, R. Pfeifer, Z. Schreter, F. Fogelman, and L. Steels, Eds. Zurich, Switzerland: Elsevier, 1989.
- [17] “Convolutional neural networks (cnns / convnets),” class notes for CS231n: Convolutional Neural Networks for Visual Recognition, Dept. of Comp. Sci., Stanford University, Palo Alto, CA, USA, spring 2017. [Online]. Available: <http://cs231n.github.io/convolutional-networks/>
- [18] J. R. Jensen, *Remote Sensing of the Environment: An Earth Resource Perspective*, 2nd ed. Upper Saddle River, NJ, USA: Pearson Prentice Hall, 2007.
- [19] E. L. Hestir, S. Khanna, M. E. Andrew, M. J. Santos, J. H. Viers, J. A. Greenberg, S. S. Rajapakse, and S. L. Ustin, “Identification of invasive vegetation using hyper-spectral remote sensing in the California Delta ecosystem,” *Remote Sensing of Environment*, vol. 112, no. 11, pp. 4034–4047, 2008.
- [20] J. F. Artiola, I. L. Pepper, and M. L. Brusseau, *Environmental Monitoring and Characterization*. San Diego, CA, USA: Elsevier Science and Technology, 2004. [Online]. Available: <https://ebookcentral.proquest.com/lib/ebook-nps/detail.action?docID=294595>

- [21] Humboldt State University. “Natural and false color composites.” 2014. Accessed Jan. 10, 2020. [Online]. Available: http://gsp.humboldt.edu/OLM/Courses/GSP_216_Online/lesson3-1/composites.html.
- [22] V. Vapnik, *Statistical learning theory*. New York, NY, USA: Wiley, 1998.
- [23] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.
- [24] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik, “Contour detection and hierarchical image segmentation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 5, pp. 898–916, 2011-05.
- [25] Camp Roberts Historical Museum. “History of the base.” 2019. Accessed Nov. 9, 2019. [Online]. Available: <https://camprobertshistoricalmuseum.com/history-of-the-base/>.
- [26] University of California, Berkeley. “Wieslander vegetation type mapping collection.” Accessed Nov. 20, 2019. [Online]. Available: <http://vtm.berkeley.edu/#/data/vegetation>.
- [27] M. Kelly, B. Allen-Diaz, and N. Kobzina, “Digitization of a historic dataset: The Wieslander California vegetation type mapping project,” *Madroño*, vol. 52, no. 3, pp. 191–201, 2005.
- [28] M. Kelly, K. Ueda, and B. Allen-Diaz, “Considerations for ecological reconstruction of historic vegetation: Analysis of the spatial uncertainties in the California vegetation type map dataset,” *Plant Ecology*, vol. 194, no. 1, pp. 37–49, 2008.
- [29] DigitalGlobe. “About DigitalGlobe.” 2019. Accessed Nov. 23, 2019. [Online]. Available: <http://www.digitalglobe.com/company/about-us>.
- [30] DigitalGlobe. “The DigitalGlobe constellation.” 2018. Westminster, CO, USA. [Online]. Available: https://dgv4-cms-production.s3.amazonaws.com/uploads/document/file/126/Constellation_Brochure_2018.pdf
- [31] DigitalGlobe. “About my DigitalGlobe.” 2019. Accessed Nov. 23, 2019. [Online]. Available: https://gcs-docs.s3.amazonaws.com/EVWHS/Introduction_Topics/Introduction.htm.
- [32] DigitalGlobe. “WorldView-2 data sheet.” 2016. Westminster, CO, USA. [Online]. Available: <https://dgv4-cms-production.s3.amazonaws.com/uploads/document/file/130/WorldView2-DS-WV2-rev2.pdf>

- [33] Insitu. "Transmission systems for communications." 2008. Bingen, WA, USA. 1.1.
- [34] Esri. "About ArcGIS Pro." 2019. Accessed Nov. 24, 2019. [Online]. Available: <https://pro.arcgis.com/en/pro-app/get-started/get-started.htm>.
- [35] Esri. "What is GIS?" 2019. Accessed Nov. 24, 2019. [Online]. Available: <https://www.esri.com/en-us/what-is-gis/overview>.
- [36] Google. "Why TensorFlow." Accessed Nov. 24, 2019. [Online]. Available: <https://www.tensorflow.org/about>.
- [37] Esri. "Export training data for deep learning." 2019. Accessed Nov. 26, 2019. [Online]. Available: <https://pro.arcgis.com/en/pro-app/tool-reference/image-analyst/export-training-data-for-deep-learning.htm>.

Initial Distribution List

1. Defense Technical Information Center
Ft. Belvoir, Virginia
2. Dudley Knox Library
Naval Postgraduate School
Monterey, California